Ref 2

(19)日本国特許庁（ＪＰ）　(12) **公表特許公報（Ａ）**　(11)特許出願公表番号

特表平8－512438

(43)公表日　平成８年(1996)12月24日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ		
Ｈ０３Ｍ	7/30		9382－5K	Ｈ０３Ｍ	7/30	Ｚ
Ｇ０９Ｃ	1/00	６１０	7259－5Ｊ	Ｇ０９Ｃ	1/00	６１０Ｄ
Ｈ０４Ｂ	1/66		4101－5Ｊ	Ｈ０４Ｂ	1/66	
Ｈ０４Ｌ	9/18		8842－5Ｊ	Ｈ０４Ｌ	9/00	６５１

審査請求　未請求　　予備審査請求　有　　（全 93 頁）

(21)出願番号　特願平6－520072
(86)(22)出願日　平成６年(1994)２月28日
(85)翻訳文提出日　平成７年(1995)９月８日
(86)国際出願番号　ＰＣＴ／ＵＳ９４／０２０８８
(87)国際公開番号　ＷＯ９４／２１０５５
(87)国際公開日　平成６年(1994)９月15日
(31)優先権主張番号　０８／０３０，７４１
(32)優先日　1993年３月12日
(33)優先権主張国　米国（ＵＳ）
(81)指定国　ＥＰ(ＡＴ，ＢＥ，ＣＨ，ＤＥ，ＤＫ，ＥＳ，ＦＲ，ＧＢ，ＧＲ，ＩＥ，ＩＴ，ＬＵ，ＭＣ，ＮＬ，ＰＴ，ＳＥ)，ＡＵ，ＣＡ，ＪＰ

(71)出願人　ザ　ジエイムス　グループ，インク
アメリカ合衆国、フロリダ州　33942、ナプレス、ノース　エアポート　ロード　125、スート　202
(72)発明者　ジェイムス、ディビッド、シー
アメリカ合衆国、フロリダ州　33937、マルコ　アイランド、パルド　イーグル　ドライブ　1085、アパートメント　Ｆ—608
(74)代理人　弁理士　木村　満　（外２名）

09/727,096

(54)【発明の名称】　データ圧縮方法

(57)【要約】
高度にランダム化されたデータを圧縮する方法を含むデータ圧縮方法が開示されている。あまりランダム化されていないデータを圧縮するためにニブル符号化、分配符号化、直接ビット符号化法が開示されている。さらに、高度にランダム化されたデータの圧縮に非常に有効なランダムデータ圧縮ルーチンも開示されている。開示された圧縮方法の全ては、ビットレベルで動作する。従って、圧縮対象データの性質や生成原因には無関係である。従って、この発明の方法は、その生成源によらず、あらゆる形式のデータに広く適用可能である。

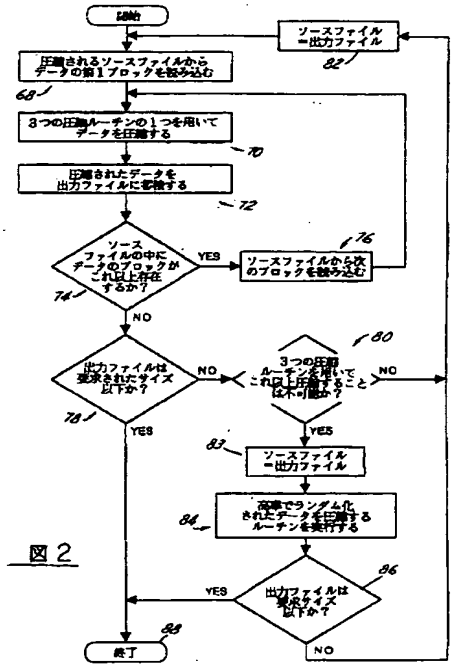


図２

【特許請求の範囲】

1．以下のステップを有し、電気的にコード化された一連のバイナリデータのストリームから冗長抽出方法、

A） 前記一連のデータからｎビットを分析し、

B） ｎビットの複数の全ての可能な状態に基づいて前記解析したｎビットの状態を決定し、

C） Bステップの結果に基づいて、前記分析したｎビットの状態を複数のコードの１つに関連付け、

D） ステップCのコードを第１と第２の部分に分解し、

E） 前記第１の部分を第１と第３の位置の少なくとも１つに割り付け、

F） 前記第２の部分を少なくとも第２の位置に割り付ける。

2． 分析している２ビットを含むｎビットをパス（通過）することを特徴とする請求項１に記載の冗長抽出方法。

3． ステップFの割り付けは、さらに、前記第１の部分を前記第１と第３の位置の少なくとも１つの内容に連結することを含むことを特徴とする請求項１に記載の冗長抽出方法。

4． ステップFの割り付けは、さらに、前記第２の部分を少なくとも前記第２の位置の内容に連結することを含むことを特徴とする請求項１に記載の冗長抽出方法。

5． D） 前記第１、第２及び第３の位置の少なくとも１つの内容を前記一連のデータに割り付け、前記A乃至Fを繰り返すステップをさらに含むことを特徴とする請求項１に記載の冗長抽出方法。

6．以下のステップを有し、一連のバイナリデータを構成する方法、

A） 前記一連のデータからｎビットを分析し、

B） ｎビットの複数の可能な状態に基づいて前記解析したｎビットの状態を決定し、

C） Bステップの結果に基づいて、第1、第２、第３の位置の少なくとも１つおける前記ｎビットの状態をコード化し、ここで、前記状態のコード化は複数

の可変長コードの生成と、前記複数のコードの少なくとも一部を少なくとも第１、第２及び第３の部分に分割し、前記コード又は前記コード部分の少なくとも１つを前記第１、第２及び第３の位置の少なくとも１つに、少なくとも一部において前記分析したｎ－ビットの所望の属性の関数として、割り付ける。

７．　前記ｎは２であることを特徴とする請求項６に記載の一連のバイナリデータを構成する方法。

８．　前記所望の特徴は前記ｎビットの値の集団を含むことを特徴とする請求項１に記載の一連のバイナリデータを構成する方法。

９．　前記複数のコードからの第１コードを前記分析したｎビットの所望の属性に関連付けるステップをさらに含む、前記第１のコードはｎ－１ビット長以下であることを特徴とする請求項６に記載の一連のバイナリデータを構成する方法。

１０．　Ｄ）　前記第１、第２及び第３の位置の少なくとも１つの内容を前記一連のバイナリデータに割り付け、前記Ａ乃至Ｆを繰り返すステップをさらに含むことを特徴とする請求項６に記載の一連のバイナリデータを構成する方法。

１１．以下のステップを有し、電気的にコード化された一連のバイナリデータのストリームから冗長データを抽出して圧縮する方法、

Ａ）　前記一連のデータからｎビットを分析し、

Ｂ）　ｎビットの複数の可能な状態に基づいて前記解析したｎビットの状態を決定し、

Ｃ）　コードをＢステップで決定された状態に関連付け、

Ｄ）　ステップＣのコードの第１の部分を第１と第３のレジスタのうちの少なくとも１つの内容に連結し、

Ｅ）　前記コードの第２の部分を第２のレジスタの内容に連結し、

Ｆ）　前記第１、第２、第３のレジスタの少なくとも１つの内容から一度にｎビットを分析し、

Ｇ）　ステップＦで分析したｎビットのそれぞれにコードを割付け、前記コードの値は前記ステップＦで分析したビットパターンに一部分で依存し、

Ｈ）　前記コードを複数の部分に分割し、前記部分を前記第１、第２、第３の

レジスタの少なくとも２つに割り付ける。

１２．以下のステップを有し、電気的にコード化された一連のバイナリデータのストリームから冗長データを繰り返して抽出して圧縮する方法、

Ａ）　前記一連のデータからｎビットを解析し、

Ｂ）　望ましいｎビットパターンの発生と、望ましくないｎビットパターンの発生を分類し、

Ｃ）　望ましいビットパターンの各発生について、第１のコード列を第１と第２のレジスタの内容に関連付け、

Ｄ）　望ましくないビットパターンの各発生について、第２のコード列を第２のレジスタの内容に関連付け、

Ｅ）　第１レジスタの内容を圧縮し、

Ｆ）　ステップＥの圧縮された内容の少なくとも一部を一連のバイナリデータに割り付け、

Ｇ）　Ａ乃至Ｆのステップを繰り返す。

１３．以下のステップを有し、電子的装置の使用によりデータを圧縮する方法、

Ａ）　データ源からデータを読み出し、

Ｂ）　前記データを使用可能なフォーマットに変換し、

Ｃ）　前記データを暗号化して、前記データ源から読み出したデータの総量減量し、

Ｄ）　暗号化されたデータを保存し、

Ｅ）　保存されたデータを読み出し、前記変換、暗号化、及び保存ステップを保存データが所望レベルに減少するまで繰り返す。

１４．　前記情報の変換は、ビットプリドミナンス法を使用して達成することを特徴とする請求項１３に記載の方法。

１５．　前記暗号化は直接ビット操作を含むことを特徴とする請求項１３に記載の方法。

１６．　前記暗号化はニブル暗号化を含むことを特徴とする請求項１３に記載の方法。

１７．　前記暗号化は分配法を含むことを特徴とする請求項１３に記載の方法。

１８．　前記分配はパックルーチンを含むことを特徴とする請求項１７に記載の方法。

１９．　前記情報は予め圧縮されたデータであることを特徴とする請求項１３に記載の方法。

２０．以下のステップを有し、ニブル暗号技術を用いてバイナリデータ列から冗長を抽出する方法、

Ａ）　バイナリデータ列からニブルを分析し、

Ｂ）　ステップＡにおいて分析されたニブルの値に応答し、複数の制御ワードの１つを７つの出力列の少なくとも１つに連結する。

２１．以下のステップを有し、分配圧縮法を用いて電気的にコード化されたバイナリデータ列からのデータを圧縮する方法、

Ａ）　前記バイナリデータ列を分析して、最も頻繁に発生するニブル値を決定し、

Ｂ）　前記データ列を所定サイズのブロックに分割し、

Ｃ）　どのブロックが最も頻繁に発生するニブルを含み、どのブロックがそうではないかを示す第１のコード化されたストリングを生成し、

Ｄ）　最も頻繁に発生するニブルを含んでいないブロックに関して、ベース２５６パッキングを用いてその内容を圧縮し、

Ｅ）　最も頻繁に発生するニブルを含んでいるブロックに関して、最も頻繁に発生するニブルが起こる位置を各ブロックについて示す第２のコード化されたストリングを生成し、

Ｆ）　第１と第２のコード化されたストリングを縮小する。

２２．以下のステップを有し、直接ビット操作法を用いて電気的にコード化されたバイナリデータ列からのデータを圧縮する方法、

Ａ）　前記バイナリデータ列を複数の入力ワードに分解し、

Ｂ）　複数のレンジを定義し、

Ｃ）　各ワードが属するレンジを決定し、

Ｄ） ステップＣで成された各決定について、各ワードをバランス値に変換し、制御ワードと分析ワードを各バランス値に付加し、バランス値と制御ワードと分析ワードはそれらに関連する入力ワードの値を個別的に決定する。

２３．以下のステップを有し、電気的にコード化され、ランダムに分配したデータ列を縮小する方法、

Ａ） 前記データ列をランダムに分配したデータの複数のブロックに分解し、

Ｂ） 前記ランダムに分配したデータのブロックの１つを選択し、

Ｃ） ステップＢで選択されたブロックを第１と第２の部分に分解し、

Ｄ） ステップＢで選択されたブロックの第１の部分内の所定のワードの発生をカウントし、

Ｅ） 前記第２の部分のデータを圧縮する。

２４． 前記圧縮はパッキングを使用することを含むことを特徴とする請求項２３に記載の方法。

２５． 前記所定のワードは、値が０の単一のビットセットと値が１の単一のビットセットから構成されるセットから選択される、ことを特徴とする請求項２３に記載の方法。

２６． 前記第１の部分と前記圧縮された第２の部分を結合し、結合されたデータに等しいデータ列を設定し、Ａ乃至Ｅのステップを繰り返す、ことを特徴とする請求項２３に記載の方法。

２７．以下のステップを有し、電気的にコード化された入力データの列から、パック化及びアンパック化が容易な電気的にコード化されたデータのコード化された可変長の列を生成する方法、

Ａ） 前記入力データ列をｎビットずつ分析し、

Ｂ） ｎビットの複数の全ての可能な状態に基づいて、各分析されるｎビットについてステータスワードをコード化し、

Ｃ） 前記コードの少なくともいくつかを第１と第２の部分に分解し、

Ｄ） 前記コードの第１の部分を第２の保存レジスタにセットし、

Ｅ） 前記コードの第２の部分を第１と第３の保存レジスタの少なくとも１つ

にセットし、

F） 第2のレジスタにおける第1のワード値の発生数が前記第1のレジスタの内容の長さを指示するように、ステータスワードのコーデングに使用する値を選択し、前記第1と第2の部分の配置を選択する。

２８． 第2のレジスタにおける第2のワード値の発生数が前記第3のレジスタの内容の長さを指示するように、ステータスワードのコーデングに使用する値を

選択し、前記第1と第2の部分の配置を選択する、ことを特徴とする請求項２７に記載の方法。

２９． 第1のレジスタにおける第2のワード値の発生数が前記第3のレジスタの内容の長さを指示するように、ステータスワードのコーデングに使用する値を選択し、前記第1と第2の部分の配置を選択する、ことを特徴とする請求項２７に記載の方法。

３０． ｎ＝２であることを特徴とする請求項２７に記載の方法。

３１． 前記コードワードは2ビット長であることを特徴とする請求項２７に記載の方法。

３２． 前記第1のワード値は値が0の単一ビットセットと値が1の単一ビットセットからなるセットの中から選択されることを特徴とする請求項２７に記載の方法。

３３． 以下の工程を有し、電気的にコード化された一連のデータを容易にパックすることができる電気的にコード化されたデータ列に分配する方法、

A） データを少なくとも第1と第2のデータ列に変換し、

B） 前記第1のデータ列を少なくとも第1のコードワードを含むようにコード化し、

C） 前記第1の列中の第1のコードワードの発生の数により前記第2のデータ列の長さを示す。

３４． 前記データを少なくとも第1、第2、及び第3のデータ列に変換し、前記第1の列を少なくとも第1と第2のコードワードを含むようにコード化し、前記第1の列中の第2のコードワードの発生数により第3のデータ列の長さを示す

、ことを特徴とする請求項３３に記載の方法。

３５．　前記データを少なくとも第１、第２、及び第３のデータ列に変換し、前記第２のデータセットを第２のコードワードを含むようにコード化し、前記第２の列中の第２のコードワードの発生数により第３のデータ列の長さを示す、ことを特徴とする請求項３３に記載の方法。

３６．　前記第１と第２の列を連結して格納する、ことを特徴とする請求項３３に記載の方法。

３７．　前記第１と第２と第３の列を連結して格納する、ことを特徴とする請求項３４に記載の方法。

３８．　前記第１と第２と第３の列を連結して格納する、ことを特徴とする請求項３５に記載の方法。

【発明の詳細な説明】

データ圧縮方法

技術分野

この発明は一般的にデータ圧縮方法に関し、特に、ディジタル型式で表現されたデータを圧縮するためにディジタル処理装置を動作させる方法に関する。

発明の背景

データ圧縮技術は、通信及びコンピュータ分野で広く用いられている。通信分野においては、受信時にデータを元の形に再構成できる圧縮データを送信することが望ましい。圧縮データの送信は、非圧縮の同一データの送信よりも送信時間が短い。コンピュータ分野では、圧縮データは非圧縮データよりも記憶領域面で有利である。したがって、固定記憶容量を持った記憶装置においては、圧縮する方がより多くのファイルを蓄積することができる。よって、データ圧縮による2つの主な利点としては、記憶容量の増大と転送時間の短縮とがある。

データ圧縮技術は損失があるタイプ（損失型）と損失がないタイプ（非損失型）の2つの大きなカテゴリーに分類される。非損失型データ圧縮技術は、圧縮や伸張（圧縮解凍）処理において、情報の損失が許されない時に使用される。損失型データ圧縮技術は非損失型データ圧縮技術よりも正確さに欠けるが、一般的に処理速度が速い。つまり、損失型データ圧縮技術は圧縮や伸張のサイクルでの処理速度のために正確さをやや犠牲にしている。損失型データ圧縮技術は、情報の損失を許容できるアプリケーション（例えば、ディジタル化したビデオデータ及びオーディオデータの送信、記憶等）の処理時に典型的に使用される。損失型データ圧縮は非損失型データ圧縮よりも圧縮率が大きく、圧縮処理速度も速い。損失型データ圧縮技術は近時、パーソナルコンピュータとその関連市場用のオーディオ及びビデオのアプリケーションの普及という観点から相当な重要性を得てきている。それ以外の大多数のアプリケーションはデータ圧縮において、非損失型デ

ータ圧縮技術を使用している。

定義によれば、非損失型データ圧縮技術はデータが圧縮や伸張のサイクルを通

過した後、データの正確な復元を保証する方法を使用している。非損失型圧縮はコンピュータで使われているディジタルデータの記憶と最も関連している。そのアプリケーションはデータベースレコード、表計算ソフト、ワープロの記憶を含む。

根本的に、全てのデータ圧縮技術は情報理論として知られる数学分野に関連し、かつこれを用いている。この数学分野は情報の表現、蓄積、通信に関する問題に関係している。

データ圧縮は冗長性との関係より、情報理論のその分野に関連している。データメッセージ中の情報が冗長ならば（その欠落がデータ内にコード化されている情報を減少させないならば）、メッセージ内にコード化されている情報を欠くことなく、データメッセージを短縮することができる。したがって、非損失型データ圧縮は、メッセージが有する情報を破壊することなく完全なまま、データメッセージのサイズを縮小する。

エントロピーは、どのくらいの情報がメッセージ内でコード化されているかを数量的に表すための用語である。高いエントロピーを持つメッセージは、同一の長さで、低いエントロピーを持つメッセージより、情報量が多い。メッセージ（通信データ）中の記号（文字）のエントロピーはそのメッセージ内において任意の文字が発生する確率の負の対数として定義される。複数ビットからなる文字の情報内容を決定するために、以下に示す2を基底とした対数を用いてエントロピーを表す。

$$E_{char}(X) = -\log_2 (\text{文字 } (X) \text{ の確率})$$

・$E_{char}(X)$ =メッセージ中の任意の文字のエントロピー

・文字 (X) の確率=そのメッセージにおいて文字 (X) が発生する確率

メッセージ全体のエントロピーはそのメッセージ中に存在するキャラクタ（又は記号）のエントロピーの単純な合計である。

エントロピーの概念は、データ圧縮技術を最適化するための指示を与える。なぜなら、その概念は、情報のどの程度のビット数がメッセージ中に実際に存在するかを論理的に決定するためである。もし、任意のメッセージ中で’Q’の文字

が1/16の確率で発生するならば、その文字が有する情報の内容は、４ビットであるということであり、’QQ’は８ビットである。’QQ’を表現するために標準８ビットのASCIIコードを使用すれば、１６ビット必要になる。エントロピーの８ビットとデータ列をコード化（エンコード）するために使用される１６ビットの違いにより、データ圧縮が可能になる。

情報理論で使用される場合、エントロピーは情報内容の相対的な尺度であり、絶対的な尺度にはなり得ない。なぜなら、キャラクタの情報の内容は任意のメッセージ中でのそのキャラクタの発生率によるためである。２つの違うメッセージにおいて、両方が少なくても１回の文字’Ｅ’の発生回数を持つならば、’Ｅ’の発生率は２つのメッセージの間で違う値になるだろう。したがって、文字’Ｅ’の情報の内容は固定された値ではなく、文字’Ｅ’の発生率に比例してメッセージ毎に変化するものである。ほとんどのデータ圧縮技術は、任意のメッセージ中で高い発生率を持つ記号（又は文字）を予測する。高い発生率を持つ記号は低い発生率を持つ記号と比較して、必然的に低い情報の内容を持ち、コード化するために必要なのビットも少ない。他の技術としては、任意の文字の発生率を確定する方法が知られている。文字データにとって、最も容易な方法は各々の文字の発生率を確定し（実験的に）、バイナリコードの長さが文字の発生率に反比例するよう、各々の文字にバイナリコードを割り当てることである。（すなわち、最も短いバイナリコードは最も頻繁に現れる文字に割り当てられる。）

辞書を基にした技術は、データの一部分又は複数の部分を走査して、どの文字又は文字列が最も頻繁に現れるかを判定する方法を用いている。文字又は文字列は辞書の中に配置されていて、文字又は文字列の発生率に反比例したコード長の所定のコードが割り当てられている。文字又は文字列はデータファイルから読み込まれ、その文字の辞書登録（見出し語）とマッチングされ、対応したコードを用いてコード化される。

近時では、データ圧縮のソフトウェアはDOSの世界において急増している。これらのソフトウェアはいくつかの欠点を有する。第１に、プログラムは典型的なディスク集中型で、その結果、プログラムの動作はディスクに読み書きするスピ

ードに規制されている。例えば、ランダムアクセス速度が18ミリ秒のハードディスクを装備した25MHZ 386 GATEWAY 2000™で作動するPKZIP™として広く知られるコンピュータ圧縮プログラムは1メガビットのASCIIファイルをその1/2のサイズに圧縮するために8.5秒かかる。データベースファイル及び表計算ソフトファイルの圧縮もほぼ同一の時間がかかるが、それらは元の2/3のサイズにしか圧縮されない。バイナリファイルが最も圧縮され、元のサイズの10～40%となる。しかし、同一の長さのASCIIファイルよりも圧縮時間が6倍かかる。

以上のデータ圧縮方法の欠点を考慮すると、より効果的及び効率的なデータ圧縮技術が必要となる。

この発明は、これまで達成されなかったデータ圧縮のレベルに達したと考えられる。既知のデータ圧縮製品はテキストファイル及びグラフィックファイルに対して50%以上の圧縮率は不可能であり、実行ファイルに対しては更に低い（約45%圧縮）。この発明を用いれば、90%のデータ圧縮レベル（特定なアプリケーションを使用すればより高い圧縮率で）が、現在のデータ圧縮製品が同一のデータを50%の圧縮率で圧縮するのと同一か又はそれ以下の時間で達成される。この発明は、ランダム（又はカオス）に出現する形式の情報から、順序付けられた情報列を分離したり、配置したりすることによって高い圧縮率を達成する。従来のデータ圧縮方法は、ランダムに（冗長なしに）配置されているように見えるデータ中の秩序（冗長）を見出できない場合が多かった。結果的に、従来の圧縮方法は、検出できない秩序を圧縮することにおいて非効率的である。

以上の説明から理解できるように、秩序付けられたデータがランダムデータから抽出されれば、そのように秩序付けられたデータは容易に圧縮できる。

この発明の第２の面は高い圧縮率を達成する性能を更に高めるものであり、再帰的に（繰り返して反復的に）その性質をデータに適用できることである。特に、この発明の方法は、ファイルを圧縮する度に、複数のパスを作成していくことが可能である。このようにして、一連の反復処理は所望の圧縮レベルが達成されるまで、実行される。

発明の概要

第１の観点において、この発明はバイナリデータ列から冗長を抽出する方法を提供する。この方法は、前記一連のデータから所定数のビットを分析（解析）し、分析したビットの複数の可能な状態に基づいて前記分析したビットの状態を決定することを含む。分析したビットの状態に基づいて、分析したビットの状態に状態コードが関連付けられる。状態コードは第１と第２の部分に分解される。ここで、第１の部分は第１と第３の位置の少なくとも１つに割り付けられ、前記第２の部分は少なくとも第２の位置に割り付けられる。

第２の観点において、この発明はバイナリデータ列を圧縮する方法を提供する。この方法は、前記一連のバイナリデータから所定数のビットを分析（解析）するステップを含む。各分析したビットの組について、ビットの状態が決定され、状態コードが、第１、第２、第３の位置の少なくとも１つに設定される。状態のコード化（コーディング）は、複数の可変長コードの生成と、前記複数のコードの少なくとも一部を少なくとも第１と第２の部分に分割することを含む。前記コード又は前記コード部分の少なくとも１つを、前記第１、第２及び第３の位置の少なくとも１つに、少なくとも一部において前記分析したｎ－ビットの所望の属性の関数として、割り付ける。

第３の観点において、この発明は、一連のバイナリデータから冗長データを繰り返して抽出して圧縮する方法を提供する。この方法は、前記一連のデータからｎビットを分析（解析）し、所望のｎビットパターンと非所望のｎビットパターンの発生を分類することを含む。第１のコード列は、所望のビットパターンの各発生により、第１と第２のレジスタの内容に連結され、第２のコードシーケンスは、非所望のビットパターンの各発生により、第２と第３のレジスタの内容に連結する。第１のレジスタの内容は圧縮され、圧縮された内容の少なくとも一部は、再び、同一の方法を用いて処理される。

さらに、第４の観点において、この発明は、一連のバイナリデータから冗長データを抽出して圧縮する方法を提供する。この方法は、前記一連のデータから所定数のビットを分析（解析）し、複数の可能な状態に基づいて前記解析したビッ

トの状態を決定する。状態にコードが関連付けられ、このコードは複数の部分に分割される。コードの第１の部分は第１と第３のレジスタの少なくとも一方の内容に連結され、コードの第２の部分は、第２のレジスタの内容に連結される。第１～第３のレジスタの内容。前記第１、第２、第３のレジスタの少なくとも１つが選択され、その内容はｎビット単位で分析される。レジスタの１つの分析されたｎビットのそれぞれにコードが割り当てられる。コード値は、前記レジスタの内容から分析されたビットのビットパターンに基づいている。コードは第１～第３のレジスタの少なくとも２つの部分に分割されている。

さらに他の観点において、この発明は、ディジタルコンピュータを用いてデータを圧縮する方法を開示する。この方法は、データ源からデータを読み出し、前記データを使用可能なフォーマットに変換し、前記データを暗号化して、前記データ源から読み出したデータの総量を減量し、暗号化されたデータを保存し、保存されたデータを読み出し、前記変換、暗号化、及び保存ステップを保存データが所望レベルに減少するまで繰り返す。

さらに、この発明は、ニブル暗号化技術を用いてバイナリデータ列から冗長を抽出する方法を開示する。この技術は、バイナリデータ列からニブルを分析し、分析されたニブルの値に応答し、複数の制御ワードの１つを７つの出力列の少なくとも１つに連結する。

さらに、第７の観点において、この発明は、分配圧縮法を用いてバイナリデータ列からのデータを圧縮する方法を提供する。この方法は、前記バイナリデータ列を分析して、最も頻繁に発生するニブル値を決定し、前記データ列を所定サイズのブロックに分割し、どのブロックが最も頻繁に発生するニブルを含み、どのブロックがそうではないかを示す第１のコード化されたストリングを生成し、最も頻繁に発生するニブルを含んでいないブロックに関して、ベース２５６パッキングを用いてその内容を圧縮し、最も頻繁に発生するニブルを含んでいないブロックに関して、最も頻繁に発生するニブルが起こる位置を各ブロックについて示す第２のコード化されたストリングを生成し、第１と第２のコード化された列を縮小する。

第8の観点において、この発明は、直接ビット操作法を用いてバイナリデータ列からのデータを圧縮する方法を開示する。この方法は、バイナリデータ列を複数の入力ワードに分解し、複数のレンジを定義し、各ワードが属するレンジを決定し、各ワードをバランス値に変換し、制御ワードと分析ワードを各バランス値に割付け、バランス値と制御ワードと分析ワードはそれらに関連する入力ワードの値を個別的に決定する。

第9の観点において、この発明はランダムに分配したデータの列を圧縮する方法を提供する。この観点は、前記データ列をランダムに分配したデータの複数のブロックに分解するステップと、前記ランダムに分配したデータのブロックの1つを選択するステップと、選択されたブロックを第1と第2の部分に分解するステップと、前記ブロックの第1の部分内の所定のワードの発生数をカウントするステップと、前記第2の部分のデータを圧縮するステップを含む。

第10の観点において、この発明は、入力データの列から、パック化及びアンパック化が容易なコード化された可変長の列を生成する方法を開示する。この方法は、前記入力データ列をnビットずつ分析するステップと、各分析されるnビットについてステータスをコード化（コーディング）するステップを備える。前記コードの少なくとも幾つかは第1と第2の部分に分解され、前記コードの第1の部分の少なくとも幾つかが第2の保存レジスタにセットされ、前記コードの第2の部分の少なくとも幾つかが第1と第3の保存レジスタの少なくとも1つにセットされる。コードワード値とコード化されたワードの部分の配置は、第2の記憶レジスタにおける第1のワード値の発生数が前記第1のレジスタの内容の長さを指示するように、選択されている。

第11の観点において、この発明は、データを容易にパックすることができるデータ列に配置する方法を開示する。この方法は、データを少なくとも第1と第2のデータ列に変換するステップと、前記第1のデータ列を少なくとも第1のコードワードを含むようにコード化するステップを含む。第2のデータ列の長さは、前記第1の列中の第1のコードワードの発生の数により示される。

この発明の他の効果とメリットは以下の実施例の説明、請求の範囲の記載、及

び図面を参考にすることにより理解できる。以下に図面を簡単に説明する。

図面の簡単な説明

図１は、ディジタルデータ源のダイアグラム図である。

図２は、この発明の圧縮方法を示す一般的なフローダイアグラムである。

図３Ａは、この発明のニブル圧縮プログラムを示す一般的なフローダイアグラムである。

図３Ｂは、この発明の分配圧縮プログラムを示す一般的なフローダイアグラムである。

図３Ｃは、この発明の直接ビット圧縮プログラムを示す一般的なフローダイアグラムである。

図４は、この発明のニブルコード化法を示す表である。

図５Ａと５Ｂは、所定の入力列に対して図４の表を適用した例を示す。

図６は、この発明のニブルコード化法により処理されたデータをデコードするための復元木を示すフローダイアグラムである。

図７Ａは図４のニブルコード化法を用いた入力列のコード化を示す表である。

図７Ｂ～７Ｅは、図６の復元木を用いて図７Ａの入力列をデコードすることを示す表である。

図８は、ニブルコード化法によりコード化された列をパッケージするための列パッケージシーケンスを示す。

図９は、図８に示される列パッケージシーケンスをデコードするためのフローダイアグラムである。

図１０は、この発明の変換／暗号化ルーチンのフローダイアグラムである。

図１１は、表２の０が優位なアルゴリズムを用いてコード化されたデータをデコードするための復元木を示す。

図１２は、表３の１が優位なアルゴリズムを用いてコード化されたデータをデコードするための復元木を示す。

図１３Ａは、表２の変換法を用いて入力列を変換する応用を示す表である。

図１３Ｂ－１３Ｇは、図１１の復元木を用いて図１３Ａの入力列のデコードを示す表である。

図１４は、この発明の変換／暗号化アルゴリズムを示すジェネラルフローチャートである。

図１５は、表４の暗号アルゴリズムによりコード化された情報をデコードする復元木を示す。

図１６Ａは、表４の暗号アルゴリズムを用いた入力列の暗号化を示す表である。

図１６Ｂ－１６Ｆは、図１５の復元木を用いた入力列の暗号化を示す表である。

図１７Ａ－１７Ｂは、所定の入力列への０優位ルーチンの適用を示す表を示す。

図１８は、単一の入力列に０優位ルーチンを適用することにより３つの出力列を生成する様子を示す図である。

図１９は、列ＴＣ１＄に図１４の変換／暗号化アルゴリズムを適用する様子を示す図である。

図２０は、図１９のグラフィカル方法を表す表形式の図である。

図２１は、列ＴＣ２＄に変換／暗号化アルゴリズムを適用する様子を示す図である。

図２２は、列ＴＥ２１２＄に変換／暗号化アルゴリズムを適用する様子を示す図である。

図２３は、列ＴＥ３１２＄に変換／暗号化アルゴリズムを適用する様子を示す図である。

図２４は、列ＴＣ３＄に変換／暗号化アルゴリズムを適用する様子を示す図である。

図２５は、所定の入力列に表４の暗号化アルゴリズムを適用する様子を示す表形式の図である。

図２６は、列ＴＣ１＄に変換／暗号化アルゴリズムを適用する様子を示す表形

式の図である。

図２７Ａは、任意の入力列に適用する変換／暗号化アルゴリズムの適用を示す図である。

図２７Ｂと２７Ｃは、図２７Ａの入力列に変換／暗号化アルゴリズムの適用を示す表形式の図である。

図２８は、図２７Ａに示される変換／暗号化アルゴリズムにより生成された列をパックするために使用する列パッケージシーケンスである。

図２９は、高度にランダム化されたデータを圧縮する方法を示すダイアグラムである。

図３０Ａと３０Ｂは、図２９の方法で使用するルックアップテーブルの例を示す。

図３１は、図２９のルーチンで圧縮されたデータを伸張（解凍）するルーチンのフローダイアグラムである。

図３２は、この発明の分配コード化法のフローダイアグラムである。

図３３Ａと３３Ｂは、図３２の方法を所定の入力列に適用した例を示す表である。

図３４は、図３２の方法で生成された列をパッケージングするために使用される列パッケージシーケンスを示す。

図３５は、この発明の直接ビットコード化法を示す表である。

図３６は、５つの所定のビット値に適用したときの図３５の直接ビットコード化法を示す。

図３７は、直接ビット操作法によりコード化されたデータをデコードするための復元木を示す。

図３８Ａ－３８Ｃは、直接ビット操作方法により繰り返して処理されるデータ列を構築する方法を示す。

図３９は、２バイト長のワード値に適用したときの直接ビットコード化法を示す第２実施例である。

好適実施例の詳細な説明

次に示す定義はこの発明を説明する目的のために提示されている。

定義

”バイアス”

バイナリデータの集合の中の０の数と比較される、同一のデータの集合の中の１の数の度合い。その度合いは比率として表現されている。例えば、任意のバイナリデータの集合が、１つの１に対して１つの０を含むならば、それは均等にバイアスされているという（すなわち50%のバイアス）。列が１より０を多く含んでいれば、それは０にバイアスされている。

”ビジット”

２進法の桁。例えば１バイトは８ビジットの長さを持つ２進数である。ビジットは１０進法のデジットに相当する。

”ビット”

Binary digIT の短縮形。０か又は１によって表現される２進法の桁。ビジットの短縮形。

”バイト”

８桁の２進数。

”エントロピー”

システムにおける秩序の尺度。情報理論の分野では、任意のメッセージに含まれた情報量の尺度。”絶対的なエントロピー”の概念は表現し難いままである。メッセージ中に特定の記号（又は記号の群）の発生率が高ければ高いほど、その記号のエントロピーは少なくなる。

”文字のエントロピー”

文字のエントロピーはその文字の発生率の負の対数として定義されている。２進数の文字の情報内容を決定するために、２を基底とした対数を用いてそのエントロピーを表現する。

$$\text{ビットの数} = -\log_2(\text{発生率})$$

メッセージ全体のエントロピーはそのメッセージ内の個々の文字のエントロピ

ーの単純な合計である。例えば、この明細書に発生する文字"e"の確率が1/16ならば、その文字が有する情報の内容は4ビットである。従って、"eeeee"という文字列が発生すれば、その文字列は合計20ビットの情報内容を持つ（各文字の確率を全ての文字に対して合計する）。

”エントロピーの限度”

任意のデータの集まりが情報の損失なしには縮小することができない、理論上最小限のサイズ。

”均等に分配されたデータ”

均等に分配されたデータはランダムに分配されたデータと同意語である。ランダムに分配されたデータにおいては、文字セット中の各文字は、その文字セット中の他の文字と同じ位メッセージ中に頻繁に現れる。

”$_nC_r$”

n個の要素から1度にr個取る場合の組合せの数を指定する数学上の表現。数学上では、$_nC_r$は次と同等である。

$$_nC_r = n! / r!(n-r)!$$

・n, r ＝一度にr個取られる事象の数

・！ ＝階乗演算子

例：8ビットの内の3ビットが1で5ビットが0の場合、いくつの違った8ビットのパターンを発生させられるか。

答え： $_8C_3 = 8!/3!(8-3)! = 56$

”ニブル”

4桁の2進数。

”パックルーチン”

アイテム又は数値を蓄積するために必要なスペースを小さくするために、2つ又はそれ以上のアイテム（又は数値）を1つの単語に結合するソフトウェアルーチン。よく知られているルーチンとしては、PACKED DECIMALルーチンがある。パックルーチンの一例を以下に示す。

例：パックルーチン使用の際は、”基本値”が定義されなければならない。基

本値は、パックされる最も大きい発生値よりも大きい値と常に等しい。例えば、次の３つのニブルをパックしたいとする。

N1 = 1011

N2 = 0001

N3 = 0111

１２は発生している最も大きいニブル値よりも大きいため、１２の基本値が設定されなければならない。

基本値 $= 11_{10} + 1_{10} = 12_{10} \Rightarrow (1100)_2$

パックされたN1,N2,N3 = N1 * (基本値)^2 + N2 * (基本値) + N3

= (1011)(1100)^2 + (0001)(1100) + (0111)

= (110001100000) + (1100) + (0111)

= (11001000011)

パックされた時の長さ(N1+N2+N3) = 11ビット

パックされない時の長さ(N1+N2+N3) = 12ビット

節約された長さ = 12 − 11ビット

=1ビット

N1,N2,N3は以下に示す方法を用いてアンパックされる。

アンパックN1 = INT (パックされたN1,N2,N3)/(基本値)^2

= INT (11001000011) / (1100)^2

= INT [1011.00100001]

= 1011

アンパックN2 = INT (パックされたN1,N2,N3)−(N1)(基本値)^2/(基本値)

= INT [(11001000011)−11000110000] /(1100)

= INT [0001.100101]

= 0001

アンパックN3 = [(パックされたN1,N2,N3)−(N1)(基本値)^2]−[(N2)(基本値)]

= [(11001000011)−(11000110000)]−[(0001)(1100)]

= 0111

ここで図１について言及すれば、この発明の方法は、全ての形式のディジタル型にコード化（エンコード）された情報を圧縮するのに適する。開示された方法はバイナリデータのパターンに焦点を当てることにより動作するので、それらはデータのソースの特性に無関係である。従って、この発明の方法は、ディジタル記録媒体に蓄積されるデータファイル５０からのデータ、音声情報５２を合成するものとして作成されている、又は作成されるであろう信号、ディジタル型で蓄積されている他の形式の情報５４、マルチメディア５６又はグラフィックファイル５８にのそれぞれの良好に動作する。更に、この発明の方法は、あらゆるタイプのディジタル記憶装置６０からの情報、又はモデム経由で送信されるディジタル

情報６２をもうまく処理する。この発明の方法は、Macintosh® ６４、PC互換機６６又は、他のディジタル処理装置等のあらゆる形式のディジタル型処理装置上で動作するのに適している。この発明の方法は専用の論理回路６５を用いて実行されることも可能である。この発明は、データの本来の本質に無関係なため、種々の形式の使用に対して効果的である。つまり、バイナリ形式で表現される全てのデータにおいて、この発明の圧縮方法は情報の損失なしに、データのサイズを小さくするのに効果的である。

次に、図２について言及すると、一般的なレベルにおいて、この明細書に開示された全ての圧縮方法は図２のフローチャートに示される。初めに、圧縮されるデータのソースが配置され、そのデータの第１ブロックがデータのソースから読み込まれる（ステップ６８）。データの第１ブロックは、開示された３つの圧縮ルーチンのうちのどれか１つを用いて圧縮される（ステップ７０）。圧縮されたデータは蓄積され（ステップ７２）、データのブロックがまだ存在するならば（ステップ７４）、それらのデータのブロックは検索され（ステップ７６）、ステップ７０と７２を通って処理される。圧縮されるデータのソースにデータのブロックが存在しないならば、圧縮されたデータファイルが要求されたサイズ以下であるかどうかを判定される（ステップ７８）。もし更にデータ圧縮が必要ならば、３つの圧縮ルーチンのどれか１つを使用して更に圧縮が可能かどうかのチェッ

クが成される（ステップ８０）。更なる圧縮が３つの圧縮ルーチンの１つを用いて可能であると判定した場合、出力ファイルをソースファイルと再定義し（ステップ８２）、再び処理ステップ６８から７４を通るように送られる。この再帰的な処理（繰返処理）は、出力ファイルが要求されたサイズ以下になるまで（ステップ７８）、又は、データの再帰が成果のないものと判断されるまで（ステップ８０）繰り返される。再帰を繰り返した後、３つの圧縮ルーチンのひとつがデータファイルのサイズの更なる圧縮に対して不可能となった場合、圧縮された出力ファイルはソースファイルとして割り当てられ（ステップ８３）、このソースファイルのデータを４番目のデータ圧縮ルーチンが処理する（ステップ８４）。４番目のデータ圧縮ルーチンがデータを処理した後（ステップ８４）、処理結果

のデータは要求されたサイズ以下かどうかをチェックされる（ステップ８６）。要求を満たしていれば、処理は終了する（ステップ８８）。更なるデータ圧縮が必要ならば、再びステップ８２を通るよう送られ、ソースファイル＝出力ファイルとしてセットされる。そして、再び処理ステップ７０から７６を通るよう送られる。７８から８６のステップの通り再帰される必要性があるなら、それらのステップを通るように送られる。

この発明の概略について説明してきたが、次にこの発明の詳細を説明する。開示された全てのデータ圧縮の重点と大きな特徴は、それらの方法論が動作する再帰的な本質に関係している。この方法論の再帰的な本質は、この方法が圧縮されるデータ上でいくつものパスを操作することを可能とすることである。実質上、この発明の圧縮方法論は前のパスで失われたデータを圧縮することが可能とされる。この再帰的な特徴は、冗長な情報の配置及び圧縮において、この方法論を非常に効果的なものとする。

この方法の大きな特徴は、ブロック７０の３つの圧縮ルーチンとブロック８４の圧縮ルーチンとを区別することである。ブロック７０で設定された３つの圧縮ルーチンは適度にランダム化された入力データを圧縮するのに効果的である。しかしながら、ブロック７０の方法は、各再帰によって、圧縮されたデータのランダムさをより高レベルにする傾向がある。従って、ブロック７０の圧縮方法は、

何度かの再帰後には、ソースファイルのサイズを更に縮小することに対して非効率的になる。従って、高いエントロピーデータの圧縮に有効なブロック８４の圧縮方法が必要となる。

ブロック８０で成された判定は、前の再帰の成功の度合いに幾分か基づいている。例えば、最後の５回の再帰の履歴は、ソースファイルを通る各パスで起こったデータ圧縮度を反映して記録される。その記録に十分な進展が見られれば、ルーチン８４を実行する必要はない。しかしながら、十分な圧縮がもはや達成されなければ、ブロック８４の方法が実行される。

ブロック８４の方法の重要な点の１つは、エントロピーが高いデータを圧縮する能力である。ブロック８４の方法を用いてソースファイルを通るパスを作成した後、結果出力ファイルはエントロピーが高くない、という事態が発生する。このような場合には、３つの圧縮方法７０の１つを通るように戻せば、更なる圧縮結果が得られる。このような場合でなければ、ブロック８４の方法はその出力ファイルを再帰することが可能である。

次に、図３Ａ、３Ｂ、３Ｃについて言及する。図２のブロック７０で言及されている３つの圧縮方法としては、ニブル圧縮９０、分配圧縮９２、直接ビット圧縮９４がある。ニブル圧縮９０と分配圧縮９２は、類似した方法で作動し、両方ともコード化（エンコード）法を使用して、選んだデータ列の中で均等でないバイアスを作成する。データ列を作成後、バイアスされた列は、ニブル圧縮９０と分配圧縮９２の圧縮エンジンの心臓部を構成する変換／暗号化ルーチン９８に渡される。バイアスデータを設定するために、ニブル圧縮９０はニブルをコード化する方法９６を用いてSTRING1$からSTRING7$までの７つのデータ列を作成する。これと対照的に、分配圧縮９２は分配コード法１００を用いてTOTCONT$とTOTMPV$の２つのバイアスされた情報の列を作成する。直接ビット圧縮９４は、ニブル暗号化９０又は分配圧縮９２に共通しない方法論を使用する。

この発明の圧縮方法論は次を以下に説明する。図２のブロック７０の中の３つの圧縮ルーチンはニブル圧縮９０、分配圧縮９２、直接ビット圧縮９４の順番で説明される。ブロック８４のランダム化されたデータの圧縮ルーチンを、３つの

方法論９０、９２、９４の各々との関連で説明する。

１．　ニブル圧縮

図２、３Ａ，４について言及すると、ニブルコード化法９６は図４の表を用いることによって最も良好に説明される。この発明のニブルコード化の方法論によっ

て実行された第１のステップは、データの第１ブロックを読み込み（ステップ６８）、ブロックをニブル毎に分析する。各ニブル値に従って、１つ又は、それ以上の予め定義されている制御ワード（ビット又はビット列）を、少なくともSTRING1$、STRING2$、STRING3$、・・・STRING7$のうちの１つの内容に関係付ける。例えば図４について言及すれば、入力ファイルの初めのニブルが３という値を持っていれば、（すなわち、バイナリ＝００１１）、STRING1$とSTRING2$の列はそれらの列の既存の内容に関係付けられた文字０を持ち、STRING3$は既存の内容に付加された文字１１を有し、STRING4$、STRING5$、STRING6$、STRING7$は従前状態から不変である。STRING1$からSTRING7$の列は入力データ列から分析される各ニブルに対して作成される。より明確にするために、ニブルコード化（エンコード）法９６によって達成しようとすることを図５Ａと５Ｂを示す。

図５Ａと５Ｂは、１６ヶのニブルからなる入力列（図５Ａと５Ｂの上部に横に沿って示されている）が図４の表中で示されているニブルコード化方法論によって処理された時のSTRING1$からSTRING7$の列を表している。図５Ａと５Ｂは、６４ビットと５０％のバイアス（０と１が均等に分布している）を持つように設定された仮定の入力列に対して、列STRING1$、STRING2$、STRING3$、STRING7$は１と０の比率が５０対５０であるということを示している。しかし、列STRING4$、STRING5$、STRING6$は元の入力列の５０％というバイアスとは違う比率でオフセットされている。入力列の５０％のバイアスが非冗長と同意ならば、入力列（図５Ａ、５Ｂ中で示されている）は６４ビット以下には縮小されない。ニブルコード化方法論はビットの数を減少させないが、非均等にバイアスされた３つの列を作成する。これらの３つの列が変換／暗号化ルーチン９８に送られる時、ルーチン９８はデータを圧縮することによって、３つのそれぞれの列の長さを短縮する

。ニブルコード化方法論において、バイアスされていない入力データ列からバイアスされたデータの列を作成する能力はこの発明の鍵となる特徴である。図５の”総ビット数”という欄から、ニブルコード化法９６は総ビット数を減少させていないということがわかる。例えば、図５Ａと５Ｂは、ニブル暗号９６は０から１５の値を持つ１６のニブルをコード化するために６６ビット必要としていること

を示している。２進数の形式でこれらの１６のニブルの値をコード化するためには６４ビット必要である。結果として、ニブルコード化法を用いれば図５Ａと５Ｂの入力列のコード化は更に２ビット必要とする。しかし、列STRING4$、STRING5$、STRING6$がバイアスされるため（同等に分配された入力データに対して）、理論上2.19、6.27、1.41ビットの節約がそれぞれに対して得られる。この表作成をサポートする計算を表１に示す。

表1

列	nCr = n!/n!(n-r)!	nCrをコード化する為に必要なビット数	元の列の長さ（ビット）	節約できるビット数（ビット）
STRING4$	8C3 = 56	5.81	8	2.19
STRING5$	5C1 = 5	2.33	5	2.67
STRING6$	3C1 = 3	1.59	3	1.41

入力列の長さの合計 = 64.00 ビット

出力列の長さの合計 = 66.00 ビット

節約分との差 => 2.19+2.67+1.41 = 6.27 ビット

59.73 ビット

２進数での節約 = 64-59.73 = 4.27 ビット 又は 4.27/64 = 6.67%

列STRING4$、STRING5$、STRING6$は変換／暗号化方法論９８を通るように送られ、ニブル圧縮９０によって達成された効果的な圧縮は6.67%に近づく。勿論、

これには、観測されるオーバーヘッドレジスタや他の”資源を守る”情報は考慮されていない。しかし、データ圧縮アプリケーション内で大量のデータを処理する時は、そのオーバーヘッドは無視してよい傾向にある。

ニブルコード化法９６の２つの重要な特徴はSTRING1$からSTRING7$の列が容易にパック又はアンパックされることと、元の入力列が７つのコード化された出力列より容易に復元（デコード）されることである。これら２つの特徴を以下に述べる。

ニブルコード化法によりコード化された列の復元

図５Ａ、５Ｂ、６について言及すると、ニブルコード化（エンコード）法９６を通って処理された入力列の復元は、次の図６の復元木（デコードツリー）に示される手順によって達せられる。どのようにして図６の復元木が使用されるかの一例を図７Ａから図７Ｅまでに示す。図７Ａにおいては、図４のニブルコード化法に４つのニブルの入力列が与えられてる。これは、図７Ａに示すSTRING1$からSTRING7$の列を構築する結果となる。その後、図６の復元木に示された処理はSTRING1$とSTRING2$の列の初めの１ビットとSTRING3$の列の初めの２ビットに適用される（図７Ｂ参照）。その結果として生じるビットの並びは、元の入力列の初めのバイトは”０００１”と等しいことを示している。これは図７Ａに示されるニブル１の値と一致する。これらのSTRING1$、STRING2$、STRING3$の列からなる４ビットは除かれ、残った右側のビットは全て左側に適切に移され、この処理が繰り返される。図７Ｃ、７Ｄ、７Ｅはどのようにして元の入力列のニブル２、３、４が図６の復元木を用いて実際に復元されるかを示している。

ニブルコード化のパッケージシーケンス

７つの列をニブルコード化法９６によりコード化する特有の方法により、データが適切なシーケンスにてパッケージされたかどうかが容易に確認される。このシーケンスを図８に示す。

図８について言及すると、ニブルコード化法９６により作成された７つの列が図８に示す方法で連結されたとすると、それらは容易に分離される。連結された

列を分離するためのアルゴリズムを図９に示す。図８、９について言及すれば、図８の連結した列を分離するためには、元の入力列におけるビット数か又はSTRING1$の元の長さが既知の値でなければならない。これは、連結した部分を分離するために、それらの列の外に蓄積しておかなければならない情報である。図９に示すように、STRING1$のビット数を知れば、連結した列からSTRING1$を分離することができる。列STRING1$の０の数がSTRING2$の長さを生成する。列STRING2$の長さがSTRING3$の最初の部分の長さを生成する。列STRING1$の１の数がSTRING4$の長さを生成する。この処理が図９に示されるように繰り返され、図８に示す１つの列を作るために連結された元の７つの列を作成する。

列STRING4$、STRING5$、STRING6$はニブルコード化法９６により復元されると、これらは変換／暗号化ルーチン９８に圧縮のため通される。

変換／暗号化ルーチン

図１０について言及すれば、変換／暗号化ルーチン９８は２段階で実行される。第１に、変換／暗号化ルーチン９８に送られた列は、その列中で文字１が文字０より優位かどうかについて判定される。１が優位ならば、その列は１が優位なルーチン１０４に送られ、処理される。１が優位でなければ（すなわち、０が優位かまたは同等）、入力列は０が優位なルーチン１０６に送られる。１が優位なルーチン１０４と０が優位なルーチン１０６は類似した方法で作動し、両方とも３つの出力列を作成する。３つの列内の少なくとも１つの列は、データ列の圧縮を行うアルゴリズム１０８によって処理される。０が優位なルーチン１０６を表２を参照して説明する。

表２

０が優位（又は同等）な場合

一度につき２ビット分析される入力列	出力列 TC1$	TC2$	TC3$
00	0	0	X
01	X	1	1
10	X	1	0
11	1	0	X

x = 変化無し

入力列が、０が優位なルーチン１０６を通って処理される時、入力列は１度に２ビットずつ分析される。分析された２ビットの値は判定され、表２に示すように、文字０又は文字１が３つの出力列— TC1$、TC2$、TC3$ —のうちの２つに付加される。よって、０が優位なルーチン１０６は入力列を処理して３つの出力列を作成する。

１が優位なルーチン１０４は、表３に示される表に基づいて３つの出力列に値を割り当てることを除いては、０が優位なルーチン１０６と類似した方法で動作する。

表３

１が優位な場合

一度につき２ビット分析される入力列	出力列 TC1$	TC2$	TC3$
00	1	0	X
01	X	1	1
10	X	1	0
11	0	0	X

x = 変化無し

表２、３に示される、０が優位な方法と１が優位な方法の特徴は、３つの出力列から入力列を容易に再構成（復元）できることである。表２を用いてコード化された情報を復元するための復元木は図１１に示される。図１１に従って、TC2$の初めのビットが０か１であるかが判別される。それが０ならば、入力列において元の２ビットが”００”か”１１”であるかを判別するためにTC1$の初めのビ

ットが判別される。TC2$の初めのビットが１ならば、入力列において元の２ビットが”１０”か”０１”であるかを判別するためにTC3$の初めのビットが判別される。TC1$、TC2$、TC3$の内容を知ることによって、容易に入力データ列を再構成することができる。

表２に示される０が優位な方法のもう１つの特徴は、TC2$の総ビット数が、入力列の全ビット数の２分の１を示すことである。従って、TC2$のビット数を知ることにより、その２倍である入力列の長さを知る。列TC2$における０の数は、TC1$におけるビットの数を決定し、TC2$における１の数はTC3$におけるビットの数を決定する。この特徴は圧縮された情報をパッケージすることに関して、後述するように非常に貴重なものである。表２に対する復元木は図１２に示され、図１１の復元木と同様に作用する。表２を用いて説明された全ての利点は、表３の方法にも同等に適用される。従って、説明は繰り返さない。一例を図１３Ａから１３Ｇに示す。その中では表２に示す方法論を用いて、入力列が３つの列に変換されている。図１３Ｂから１３Ｇは、図１１の復元木を用いて、３つの列がどのようにして元の入力列に再構成されるかを示している。

図１３Ａから１３Ｇについて言及すれば、入力列は表２の変換方法論を用いて処理され、TC1$、TC2$、TC3$の３つの列を作成する（図１３Ａ参照）。図１１の復元木はTC2$の初めのビットに適用される。初めのビットが０なので、図１１の復元木は、TC1$の値を判別するために、TC1$の初めのビットを見るように指示している。TC1$とTC2$に対する”００”という値は、”００”の値の入力列を作成する（図１３Ｂ参照）。これらのTC1$とTC2$の２つのビットは除かれ、それらの列の残りが左に１ビットずつ移され、残りのビットがなくなるまでこの処理が繰り返される（図１３Ｃ～１３Ｇ参照）。０が優位なルーチン１０６と１が優位なルーチン１０４のどちらもデータを圧縮しないことに留意することが重要である。TC1$、TC2$、TC3$の長さを合わせたものは入力データの元の長さと常に等しい。しかし、ルーチン１０４と１０６は０にバイアスされたTC1$に対して効果的であるし又特定のビットパターンの入力列に依存するいくつかの場合において、０にバイアスされたTC3$に対して効果的でもある。一般に、TC2$の中のデータは均

等バイアスの傾向が強い。

図１０、１４について言及すれば、変換／暗号化アルゴリズムは決められた順序による０が優位なルーチンと暗号化ルーチンの実行から構成される。０が優位なルーチンについては検討済みなので、更なる説明は不要である。しかし、変換／暗号化アルゴリズム１０８について十分に理解される前に暗号化ルーチンを説明しなければならない。以下に暗号化アルゴリズムについて説明する。

暗号化アルゴリズム

暗号化アルゴリズム処理は表４との提携により最も深く理解される。

表4

暗号化アルゴリズム				
一度につき２ビット分析される入力列	出力列	TE1$	TE2$	TE3$
00		0	X	X
01		1	0	0
10		1	1	X
11		1	0	1
x = 変化無し				

表４の使用法は表２と表３が使用された方法と非常に類似している。暗号化アルゴリズムは一度に２ビットの入力列を分析することにより作動する。分析された２ビットの値は、１ビットが３つの出力列の１つに連結されるかどうかを判定する。また、分析された２ビットの値は、連結されるビットの値も決定する。入力列に暗号化アルゴリズムを適用した結果が出力列TE1$、TE2$、TE3$の生成である。３つの出力列が既知であれば、入力列は容易に再構成される。この再構成は図１５の復元木を適用する。図１５の復元木の適用は以前に説明した図１１、１２の復元木が適用される方法と同一であり、それに応じた図１５の復元木の適用の詳細な説明は不必要である。

表４の暗号化アルゴリズムの適用の一例を１６Ａから１６Ｆに示す。図１６では１０ビットの入力列が、出力列TE1$、TE2$、TE3$を作成する表４の暗号化アルゴリズムによって処理される。これらの３つの列は図１５の探索木に従って処理

され、１０ビットの入力列を正確に再生する（図１６Ｂから図１６Ｆ参照）。表４の暗号化アルゴリズムの重要な面は、０にバイアスされたデータを圧縮する能力である。例えば、図１６Ａにおいて入力列は大きく０にバイアスされる（８０％の０）。これは暗号化アルゴリズムに対して理想的なバイアス配列であり、このアルゴリズムは１０ビットの列を７ビットに短縮する（３０％の短縮）。この例より、入力列が暗号化アルゴリズムによって処理される場合、入力列が０にバイアスされていれば、結果データの３列が圧縮されることは容易に理解される。暗号化アルゴリズム（ニブルコード化のような）と提携して作動するルーチンを０に大きくバイアスされたデータの列に適用することができれば、暗号化アルゴリズムはデータ圧縮に対して非常に効果的である。表４の暗号化アルゴリズムより、そのアルゴリズムが１にバイアスされた入力列を処理すれば、そのアルゴリズムは入力データを増大させ、データの全長を増大させてしまうことは明らかである。例えば、図１６の１０ビットの入力列の１の補数が暗号化アルゴリズムによって処理される場合、列TE1$、TE2$、TE3$の長さの合計は１５ビットとなる（１５０％に増大）。データの増大は圧縮ルーチンと関連する好ましくない特徴であるため、十分なソフトウェアトラップはそのような状況の発生を防ぐために設置されなければならない。そのようなトラップは図２のフローチャートのブロック８０と提携して以前に説明された。

暗号化及び変換アルゴリズムについて個別に説明したので、この発明の再帰的な本質の一面を示す、暗号化方法論と変換方法論とを結合した複数のアプリケーションによるいくつかの方法を以下に示す。任意のデータセットに繰り返し適用される暗号化／変換アルゴリズム１０８の能力は、この発明の圧縮方法論が力強いデータ圧縮”エンジン”を形成することを可能にする。任意のデータセットを再帰する変換／暗号化アルゴリズム間の相互作用は次に示す例によって最も容易に理解される。

変換／暗号化アルゴリズム使用時に用いられる再帰的方法論

図３Ａ、５Ａ、５Ｂ、１０、１７Ａ、1･7Ｂについて言及すると、ニブルコード化法９６は、STRING1$、STRING2$、・・・STRING7$を作成する仮定の入力デー

タに適用される。この仮定の場合において、STRING4$が大きく０にバイアスされているため、その列が変換／暗号化ルーチン９８によって処理される理想的な候補を形成することを想定してみよう。この仮定の列は選ばれ、変換／暗号化ルーチン９８を通されて、０が優位なルーチン１０６を通される。図１７Ａ、１７Ｂは仮定の入力列STRING4$における変換アルゴリズムの動作を表現している（以下、変換は表２の０が優位なルーチンと同義語である）。図１７の右側の欄から明らかにされるように、仮定の入力列は６４ビットで０に７８％強バイアスされている。入力列上で作動する０が優位なルーチンは、３つの出力列TC1$、TC2$、TC3$を作成する。以前に説明したとおり、０が優位なルーチンは圧縮アルゴリズムではないので、データの全ビット数を減少させない。しかし、そのルーチンは少なくとも１つは０にバイアスされた列を作成するよう動作する。例においては、０にバイアスされた列はTC1$であり、９５％のバイアスを帯びている。図１８について言及すれば、仮定の入力列STRING4$を含む６４ビットを処理し、３つの出力列TC1$、TC2$、TC3$を作成する。

図１９、２０について言及すれば、TC1$はいくつかの出力列の中で最も高く０にバイアスされているので、変換／暗号化アルゴリズム１０８を通るように処理されるのに３つの列の中で最適なものとなる。アルゴリズム１０８を通るTC1$の処理は、図１９に示され又図２０に解析的に示されている。アルゴリズム１０８を通って列を処理するために使用される１つのその方法は、初めに入力列を表４の暗号アルゴリズムを通して処理し、データ圧縮の成果があるかどうかの判定をする。例えば、暗号化アルゴリズム１０８を通るTC1$の初めのパスの間で、３つの列TE111$（長さ１０ビット）、TE211$（長さ１ビット）、TE311$（長さ１ビット）が作成される。これはTC1$の全長を８ビットだけ減少させる。長さの減少が起こらなければ、又はデータの増大が起こったら、データ列TC1$を暗号化（コード化）することは有益ではなく、表２の変換アルゴリズムを適用する。１つの可能な変換／暗号化アルゴリズム１０８はまず初めに暗号化アルゴリズムを用いて入力列を処理し、データ圧縮の成果があるかないかを判定する。圧縮の成果がなければ、列は変換される（０が優位な方法論を使用して）。この特別な方法を”

処理及び判定”方法として呼ぶこととする。”処理及び判定”方法を出力列に適用し続けることにより、各出力列TE111$、TE211$、TE311$を入力列にして、暗号化アルゴリズムを通して処理する。図１９に開示されるように、TE111$の暗号化はデータを４ビット縮小させる。列TE211$に暗号図を適用すると、暗号化ルーチンは単一ビット列上で動作することがわかり、この”レッグ”の更なる処理を控える。いくつかの例において、暗号化アルゴリズムの動作は列の増大を引き起こしている。これらの場合は暗号化（コード化）せず、そのレッグの更なる処理を控える。TE311$は１ビットなので、それに対する更なる処理は控える。

再び、”処理及び判定”方法をTE111$の暗号化により作成された３つの列に適用すると、TE121$の暗号化はデータを２ビット縮小させる。この特定の暗号化ルーチンのアプリケーションにおいて、列TE121$はビット数が奇数なので、１ビット（図１９参照）の残余ビット（残り）がある。これは、奇数ビット数の列が暗号化ルーチンか又は変換ルーチンで処理される場合、常に起こる。”処理及び判定”方法の再帰的な列TC1$への適用において説明した工程を図２０に示す。再帰的な”処理及び判定”方法の列TC1$への適用における最終結果はTC1$を８ビットの長さに縮小、又は６０％の縮小である。

図１８、２１について言及すると、TC1$の圧縮時に用いられた方法と同じものをTC2$に適用する。特に、まず暗号化アルゴリズムは圧縮成果があるかないかを判定するため適用される。TC2$の例では、３２ビットの列TC2$への暗号化ルーチンの適用は、圧縮の成果がない（暗号化ルーチンで作成された３つの列の全長さは入力列の長さと等しい）。従って、TC2$を暗号化せず、変換アルゴリズムを適用する。前述のように、変換アルゴリズムは圧縮アルゴリズムではなく、出力列の少なくとも１つが０のバイアスを帯びるようにするツールとして役立つ。列TC2$の変換は３つの出力列TE112$、TE212$、TE312$を作成する。データ圧縮が可能かどうかを判別する”処理及び判定”方法を使用して、順番にこれら３つの列は判定される。初めに、列TE112$は、連結した長さがTE112$の長さと同一な３つの列に暗号化される。この点では、２つの”変換”が並んで実行される（元の列STRING4$の変換と列TC2$の変換）。結果としての３つの列は圧縮の成果があるかど

うかを判定するために暗号化される。”処理及び判定”方法の手順に従って、初めに列TE122$の暗号化を試行するが、これは１ビットの増加を生じさせる。列TE122$の追跡は成果がないと想定し、変換アルゴリズムをその列に適用すると、４ビットの列を更に圧縮することは不可能なことに気付く。よって、列TE122$を更に圧縮することは不可能である。

暗号化された列TE122$が暗号化されると（図２１参照）、１ビット増加することに気付くことが重要である。盲目的に暗号化アルゴリズムを適用することは、データの増加を引き起こす可能性がある。列TE212$の分析と列TE312$の分析は図２２、２３に各々示される。全ての縮小が合計されると、”処理及び判定”方法は列TC2$長さの７ビット又は約２１％の縮小という結果となる。図２４に示すように、”処理及び判定”解析を列TC3$に適用すると、列TC3$の長さにおいて３ビットの縮小が得られる（又は２５％）。以上のことより、変換／暗号化アルゴリズム１０８が”処理及び判定”方法の形式を取った時、初期の列STRING4$の有力で総合的な圧縮が得られる。この例において、列STRING4$を２２ビット又は３４％近くだけ縮小させることができる。

変換／暗号化アルゴリズム１０８の実行時に”処理及び判定”方法を用いることは十分なデータ圧縮を作り出すけれども、その方法の実行は、単純ですっきりしたプログラミング技術には成り得ない。これは非常に本質的なことである。なぜなら、”処理及び判定”方法は、各方式の処理での圧縮結果を最適化するように設計されているからである。その方法はデータを高速に圧縮するけれども、暗号化又は変換されたデータを復元する時、全ての分岐において成された判定（すなわち、変換又はコード化するために）は再呼び出しされる。その情報を記録する技術は容易であるが、それはソフトウェアに望ましくない複雑さを与える。加えて、各ノードでのその最適化は不必要であると考えられている。なぜなら、圧縮可能なデータが、変換／暗号化ルーチン９８の初めの実行によって圧縮されないならば、そのデータは次のパスで圧縮ルーチン７０（図２参照）によって繰り返して処理される時、データの次に続くパスの中の１つで処理される。よって、再帰はこの発明において２つのレベルで発生する。第１のレベルでは、再帰は変

換／暗号化アルゴリズムによって実行される複数の変換／暗号化アプリケーションにおいて発生する。第２のレベルでは、再帰はデータを必要なサイズ以下に縮小する圧縮ルーチン７８の複数のアプリケーションによって、発生する。この見解によれば、変換／暗号化アルゴリズム１０８を用いた”処理及び判定”方法は非最適化方式より優れた圧縮結果を達成するとは考えられていない（”処理及び判定”方法は高速に結果を出すけれども）。より時間をかければ、簡易化された変換／暗号化アルゴリズムが使用される。簡易なアルゴリズムを以下に開示する。

この簡易化した技術を説明することにおいて、図１９から図２４に図示された例の説明を再検討することは有益である。これらの図より、最も大きい圧縮成果が最左方の分岐から集められているということが一般的に観察される。この傾向はほとんど全てのレベルで観察される。例えば、列TC1$、TC2$、TC3$の中で最も大きい総合的な圧縮は列TC1$による、６０％の縮小だった。TC1$は列STRING4$から作成された最も左方の列である。次に、一般的に圧縮／暗号化アルゴリズムによる圧縮時において、最も右側の列が中心の列よりも成果があることが観察される。これらの傾向は驚くべきものではない。前述のように、変換ルーチンは可能な限り０を多く含むTC1$（最も左方の分岐）を”ロード”するように設計されている。また、TC2$は５０％のバイアス度に近づく傾向にある。最も左方の分岐は、より多くの０を含む傾向にあるため、暗号化アルゴリズムはこれらの列に対して最も効果的となる。

この考え方から、１つのより簡易な変換／暗号化アルゴリズムは分析実行時、最も左方の枝を見る。例えば、変換してコード化する判定を常に実行できるが、変換後、最も左方の枝だけコード化することもできる。次に、第１の変換をして、第１の変換の最も左方の枝について第２の変換をした後、暗号化（コード化）をする。しかし、第２の変換の最も左方の枝だけを暗号化することもできる。より洗練された方法は、ブロック毎のデータの分析と、その分析に基づいて最適化された変換／暗号アルゴリズムを判定しデータ分配パターンを選ぶことである。各最適化されたアルゴリズムはあるワード値を与えられ、そのワード値は入って

くるデータの最初のワードとして蓄積される。この目的の為に２ビットのワードを作成することにより、４つの最適化されたアルゴリズムが有益である。３ビット用いることにより、８つの最適化されたアルゴリズムがコード化される。

列STRING5$、STRING6$がSTRING4$に対して示した方法と同様にコード化／変換アルゴリズムを通ることは明白である。列STRING5$、STRING6$は１より０を多く含んでいるので、圧縮／暗号化アルゴリズム１０８が、これらの列の長さの圧縮に対して成果があることが期待される。ニブル圧縮ルーチン９０の１つのパスの完了により、４つの圧縮されない列（STRING1$、STRING2、$STRING3$、STRING7$）と３つの圧縮される列（STRING4$、STRING5$、STRING6$）が作成される。圧縮された列は１連の可変長ビット列の数により表され、その各列は１つ又はそれ以上の暗号化アルゴリズム及び変換アルゴリズムの適用することによって作成される。ニブル圧縮ルーチン９０の１つのパスで作成される種々の列と列の部分が、どのようにして後の元の入力列の再構成に使用されるようにパックされるかの一例を次に示す。

図３Ａ、８、９、２５について言及すれば、この例の目的に対して、ニブルコード化法９６がSTRING4$を（この列のみ）圧縮のために変換／暗号化ルーチン９８に通したと想定しよう。そして、変換／暗号化ルーチンに通される１つの暗号化データを単純に処理する非常に簡易な暗号化／変換アルゴリズムを想定しよう。図２５において、暗号化アルゴリズムが３２ビットの入力列を処理し、TE1$、TE2$、TE3$から成る３３ビットの出力列を作成している。傍注として、入力列において十分な０のバイアスがなければ、暗号化ルーチンがデータを１ビットだけ増大させる。マイナスの結果が発生するが、この出力列の圧縮の例は不変であり、弱まりもしない。

図２５において、出力列TE1$の総ビット数（１６ビット）が入力列の総ビット数の１／２となっている。これは入力列が偶数ビット数である限り常に起こる現象である。入力列が奇数ビット数の場合は残余が生じる。この場合を図２６の例に示す。

TE1$における１の数が列TE2$の総ビット数を決定する。更にTE2$の０の総数が

TE3$の総ビット数を決定する。３つの列をTE1$/TE2$/TE3$の順序で連結すると、元の入力列の長さを知ることにより、それらは容易に互いから分離できる。例えば、元の入力列が３２ビット長であることが既知の場合において、まず初めの３２ビットを、連結された列から取り出す。これがTE1$の列になる。列TE1$の１の数を計数（カウント）する。その数から、変数TE2$を形成するために、連結された列から次に取り出すべきビットの数が決定される。次に、列TE2$における０の数が計数され、それによりTE3$に割り当てられるビットの数が決定する。この例により、暗号化ルーチンによって作成された３つの出力列は他の列との関連で特有に定義されており、それらの列は所定の順番で連結されていて、その結果、元の入力列の長さを知ることにより、個々に識別され、連結から分離される。従って、この発明の暗号化ルーチンは０のバイアスデータの圧縮のための力強いツールなだけでなく、連結が容易に解除される出力列を作成する。

暗号化ルーチンが図２５の連結された出力列を作成すると、この列を図８の列のパッケージシーケンスに示されるSTRING4$の代わりとする。従って、図９の復元木の実行時、STRING4$の０又は１の数が必要になった適当な時点で、コード化された列STRING4$はその構成要素の列（TE1$、TE2$、TE3$）に分離され、図１５の復元木に従って復元される。これは元の入力列の再構成であり、図９の復元木はニブルコード化法で用いられた列の連続鎖の復元を実行できる。

図２６について言及すれば、第２の例では、第１の例で用いられたものより複雑なコード化／変換アルゴリズムが用いられる。第２の例では変換ステップを用い、そして暗号化ステップを用いる。この第２の例を図２６を参照して説明する。

まず初めに、３２ビットの入力列を変換ルーチンを使用して処理し、それによりTC1$、TC2$、TC3$を作成する。TC2$の総ビット数は入力列の総ビット数の１／２である。これは入力列が奇数のビット数でなければ、常に起こる状況で、入力列が奇数のビット数の場合、残余が生じる。列TC2$での１の数を計数することにより、TC3$の総ビット数を得る。更に、列TC2$での０ビットの数はTC1$の総ビット数を直接表す。この例において、TC1$を取得し、暗号化の入力列として使用す

る。TC1$は奇数のビット数なので、値が０の残余ビットが生じる。この残余ビットは記録され、暗号化アルゴリズムは、通常通りTC1$を暗号化（コード化）し、残余ビットが存在しないものとして扱うよう実行される。最終的な出力列を図２６に示す順序で作成すれば、元の３２ビットの入力列は図２５に示されるものと類似した技術を用いて容易に再構成される。

定義によって、元の入力列（３２ビット）の長さは既知のもので、そのビット数の１／２を出力列から取り出し、TC2$を生成する。TC2$の１の数は、出力列から取り出すTC3$の長さを決定する。TC2$の０の数を計数し、その数を半分に割ることによって（小数部は除く）、TE1$の総ビット数が決定される。TC1$は奇数のビット数なので、TE1$に割り当てられた４ビットの後に残余ビットが存在する。

その残余ビットを除去し、列TE1$の最後に連結する。TE1$の１の総数によりTE2$の総ビット数が決定され、TE1$の０の総数によりTE3$の総ビット数が決定される。TE1$、TE2$、TE3$はTC1$を再構成するために図１５の復元木を用いて結合され、その後TC2$、TC3$、TC1$は元の入力列を再構成するために、図７の復元木に従って結合される。

図２７Ａ、２７Ｂ、２７Ｃ、２８について言及すれば、変換／コード化アルゴリズムの第３の例を説明する。この例では、変換／変換／コード化／コード化技術が用いられる。種々の変換／コード化ステップが図２７Ａに示される”ツリー”ダイアグラムに従って元の入力列に適用される。第１と第２の例を用いて説明された解析は、この第３の例に適用される解析と同一のものである。変換／変換／コード化／コード化アルゴリズムのアプリケーションにおいて作成される全ての列及び列の構成要素は、１つの出力列を形成する図２８の方法において連結される。元の４６ビットの入力列を再構成する時、出力列は以下に示す方法を用いてその構成要素列に分離される。まず初めに、定義により入力列の長さは既知なので、その入力列の長さの（１／２）×（１／２）（すなわち１／４）と等しい列Ｇ５の長さも既知である。入力列は４６ビットなので、列Ｇ５は１１．５ビットである。少数部分を除去するこにより、ＧＳ５の長さは１１ビットとなる。出力列から初めの１１ビットを取り出し、その中の０を計数することにより、列ＧＧＳ

４の長さを決定する。処理は以下のように進む。

・　列ＧＧＳ４の１を計数することにより列ＧＧＳ５の長さを決定する。

・　列ＧＧＳ５の０の数を計数することにより列ＧＧＳ６の長さを決定する。

・　図１５の復元木に列ＧＧＳ４、ＧＧＳ５、ＧＧＳ６を適用することにより、列ＧＧＳ４を再構成する。

・　図１１の復元木に列ＧＧＳ５、ＧＧＳ６、ＧＧＳ４を適用することにより、

列ＳＯＮ２を再構成する。

・　列ＳＯＮ２はビット数が奇数なので、終端の残余ビットを取り出す。

・　ＳＯＮ２の０の数を計数することにより列ＳＯＮ１の長さを決定する。

・　ＳＯＮ１の長さを１／２に分割することにより列ＧＳ２の長さを決定する。

・　ＧＳ２の１の数を計数することにより列ＧＳ３の長さを決定する。

・　ＧＳ２の０の数を計数し、その数を１／２に分割することにより列ＧＧＳ１の長さを決定する。

・　図１５の復元木に従って列ＧＧＳ１、ＧＧＳ２、ＧＧＳ３を結合することにより、列ＧＳ１を再構成する。

・　図１１の復元木に従って列ＧＳ１、ＧＳ２、ＧＳ３を結合することにより、列ＳＯＮ１を再構成する。

・　図１１の復元木に従って列ＳＯＮ１、ＳＯＮ２、ＳＯＮ３を結合することにより、元の入力列を再構成する。

次に図３Ａについて言及すると、この発明のニブル圧縮方法論９０を要約すれば、ニブル圧縮方法論９０はニブルコード化法９６と変換／暗号化ルーチン９８から構成されるという説明がされている。ニブルコード化法は７列のデータを作成するのに効果的である。その列のいくつかは０にバイアスされている。変換／暗号化ルーチン９８はニブルコード化法９６によって作成され、０にバイアスされた列の圧縮に効果的である。種々のコード化技術がニブルコード化法９６と変換／暗号化ルーチン９８の両方において使用され、それらによって作成された列

は、隣接する列の長さの情報を持つ。このコード化技術は元の入力データの容易な記憶、検索、再構成を考慮している。

図２、３Ａについて言及すれば、ソースファイルにおける全ての入力データがニブル圧縮９０によって処理された後、出力ファイルは要求されたサイズ以下かどうかを判定するためにチェックが行われる（ステップ７８）。ソースファイルを通る１つのパスにおいて十分圧縮が実行されていれば、プログラムは終了する（ステップ８８）。しかし、多くのアプリケーションにおいて、データを繰り返して処理した場合にのみ（ステップ８０、８２）、十分な圧縮が達せられるという場合がある。しかし、複数の再帰を圧縮対象データに適用する場合において、３つの圧縮ルーチンの内の１つがデータのサイズの更なる縮小においてもはや効果的でないという点に達する。この時点では、更なる再帰は成果がなく、ある場合においては、データを増大させてしまうかもしれない（暗号化アルゴリズムが十分に０で重み付けされていないデータに適用された場合について、以前に説明したように）。圧縮ルーチン７０を更に反復しても成果がない場合、ランダム化されたデータの圧縮に対して非常に効果のあるルーチン８４を用いる。このランダム化されたデータの圧縮ルーチン８４を次に説明する。

図２、３Ａ，３Ｂ，３Ｃ，２９について言及すると、３つの圧縮ルーチンのどれか１つを用いて複数回反復して処理を行うと、出力ファイル中の１と０の分配が、高い率でランダム化し始める（すなわち、０と１の分配が、出力ファイルから取り出されるどんなサイズのサンプルに対しても５０％に近づく）。この時点において、圧縮ルーチン７０はまだ圧縮能力はあるが、効率という点で不利である。

図２９の圧縮ルーチンは、高率でランダム化されたデータの圧縮に関連した特有の問題を処理するのに適している。図２９について言及すれば、この特有の方法論の圧縮ルーチンは、高い率でランダム化されたデータを含むソースファイルからデータの初めのブロックの読み込みを行う（ステップ１１２）。この例のため、データの１ブロックサイズが１００ビットであると仮定しよう。次に、そのブロックの所定の部分における０の数が計数される（ステップ１１４）。例えば

、１００ビットの内の初めの８０ビットにおいて発生する０の数を計数する。計数された０の数と、完全なランダム化（すなわち、５０％の０と５０％の１）が成されているという仮定の基に、どの程度の０がブロックの残りの部分に残るかがわかる。これを知ることにより、ルックアップテーブルはコード化され、その中では存在する可能性がある全てのビットパターンが各々のアドレスに連結されている。ブロックの残りの部分は除去され、ルックアップテーブルのアドレスが代わりに配置される（ステップ１１６）。データの初めの部分とコード化された部分は蓄積され（ステップ１１８）、次のブロックがソースファイルから読み込まれ（ステップ１２２）、１１８を通るように処理される（ステップ１１４）。

ランダム化されたデータの列の圧縮は不可能であることは広く明言されている。この誤信は、多数のランダム化されたデータセットに対して、可能なビットパターンの組み合わせを計算することは困難である事実を根拠にした部分にある。例えば、１００ヶの１と１００ヶの０が２００ビット長の１ブロック内で分配されれば、

$$_{200}C_{100} = 9.055 \times 10^{58}$$

の組み合わせが可能である。その数が取り扱いにくいことは明確であり、これがランダム化されたデータ列は圧縮不可能であると言われる始まりである。この説明における、この発明の圧縮法の重要な点は、そのような大量のデータを扱わず、より小さな部分を処理するところにある。ランダム化されたデータの圧縮ルーチン８４の一例を図３０Ａ、３０Ｂを用いて説明する。

この例において、ブロックサイズは２００ビット長とし、ブロック毎にデータの最後の５％（１０ビット）だけ圧縮する。第１段階は初めの１９０ビットにおいて発生する０の数を計数することである。これを知ることにより、０と１が完全に均等に分配されていると仮定した場合に、いくつの０が最後の１０ビットに

発生するはずであるかを知る。最後の１０ビットの内０が９ヶなら、ルックアップテーブル１を参照する（３０Ａ参照）。ルックアップテーブル１は一意のルックアップテーブルアドレスを、可能な各ビットパターンに関連付ける。適切なルックアップテーブルアドレスが選択され、初めの１９０ビットに連結される。次

の列全体が蓄積され、次の２００ビットの列が読み込まれ、同様に処理される。初めの１９０ビットの除去後、残りの１０ビットが0が８ヶのビットパターンであれば、ルックアップテーブル２（３０Ｂ参照）が、前の１９０ビットに連結するアドレスの判定に用いられる。

種々のビットパターンに対するアドレスをコード化するこのルックアップテーブル技術を用いることによって、１０．０ビットから２．０ビットの範囲で変動するビットセービングが可能である（表５参照）。

表5

最後の10ビットにおける0の残数	nCr	アドレスをコード化する為に必要なビット数	節約できるビット数
0	0	0	10−0 = 10
1	10	3.34	10−3.34 = 6.66
2	45	5.5	10−5.5 = 4.5
3	120	7.0	10−7.0 = 3.0
4	210	7.8	10−7.8 = 2.2
5	252	8.0	10−8.0 = 2.0
6	210	7.8	10−7.8 = 2.2
7	120	7.0	10−7.0 = 3.0
8	45	5.5	10−5.5 = 4.5
9	10	3.34	10−3.34 = 6.66
10	0	0	10−0 = 10

セービング（節約量）を増加させるには、非圧縮のブロックの部分１１４を減少させ、ステップ１１６で処理される残りの部分をより大きくする。例えば、１ブロック２００ビットのうちの最後の２０ビットを用いれば（最後の１０ビットの代わりに）、最悪の場合において必要とされる最大ビット数は

$$_{20}C_{10} = 184,755 \Rightarrow 17.6 \text{（コード化するために必要なビット）}$$

であり、最悪な場合のセービングは

$$20 - 17.6 = 2.5 \text{ビット}$$

である。

図２について言及すれば、ランダム化されたデータ圧縮ルーチン８４が全ソースファイルを１回通った後、圧縮されたデータが要求されたサイズか又はそれ以

下かどうかを判別するためのチェック８６が再び成される。条件が満たされていれば、プログラムは終了する（ステップ８８）。条件が満たされていなければ、データは３つの圧縮ルーチンの１つ７０を通るように送り戻されるか、又は圧縮ルーチン８４を通るように送り戻される。データをルーチン７０とルーチン８４のどちらに送り返すべきかを判定するために用いることが可能な１つの方法は、単純にファイルのバイアスを調べることである。ファイルのバイアス率が予め設定していたレベル以下だった場合、ルーチン８４を用いると圧縮効果はより大きくなりやすい。バイアス率が予め設定していたレベル以上だった場合、圧縮ルーチン７０がルーチン８４より優れた結果を出しやすい。

ランダム化されたデータ圧縮ルーチン８４を完全にランダム化されたファイルを用いた場合でだけ説明したが、このルーチンは、各サンプルされたブロックに１ビットを単純に加えることによって、完全にランダム化されていないファイルにも適用できる。例えば、このビットに１を設定するならば、このブロックは完全にランダム化されておらず、それゆえルーチン８４による処理は不可能であるということを意味する。このビットに０を設定するならば、そのブロックに含まれるデータをルーチン８４により処理することは可能であるということを意味する。

図３１について言及すれば、ランダム化されたデータ圧縮ルーチン８４により圧縮されたデータの伸張（圧縮解凍）において、データの初めのセグメントが読み出される。次に、非圧縮部分の０の数が計数される（ステップ１２４、１２６）。この数を知ることにより、圧縮された部分を伸張すれば、どれだけの０が表れる

かを計算できる。どれだけの０が伸張される部分の中に存在するかを知ることにより、アドレスはどれくらいの長さか、そして圧縮された部分はどれくらいの長さかをルックアップテーブル技術を用いて知る（ステップ１２８）。次に適当なルックアップテーブルを参照し（ステップ１３０）、コード化されたデータ部分を適当なビットパターンに置換する（ステップ１３２）。このシーケンスはソースファイルの全内容が処理されるまで（ステップ１３４）、繰り返される（ステ

ップ１３８）。

図２について言及すれば、ランダム化されたデータ圧縮ルーチン８４が高率で又は完全にランダム化されたデータを圧縮するためにどのように動作するかを説明した。ファイル及びデータ圧縮を発生させるパス上を通る複数のパスを作成するために、どのようにルーチン８４を用いるかについても説明した。加えて、ランダム化されたデータ圧縮ルーチン８４と圧縮ルーチン７０間の相互作用が示されていて、それはデータを特徴付ける０のバイアスの度合いにかまわず、データ圧縮に効果的である。

図２、３Ａ、３Ｂについて言及すれば、すでに説明したとおり、ニブル圧縮９０は３つの圧縮ルーチンの内の１つでブロック７０にて用いられる。次に、データ圧縮に使用するためにブロック７０で用いられる第２の圧縮ルーチンについて示す。この第２のルーチンは分配圧縮ルーチンであり、それは図３Ｂに示される。

２．　分配圧縮

図３Ｂについて言及すれば、分配圧縮９２はニブル圧縮９０（図３Ａ参照）を用いて示したのと同様な一般的な形式をとる。特に、ニブル圧縮９０と同様に、分配圧縮は２工程の方法を用いる。すなわち、初めの未加工のデータはデータの複数の列作成のため分配コード化法により処理される。これらの列のいくつかは圧縮データ作成のため変換／暗号化ルーチン９８によって処理される。変換／暗号化ルーチン９８は図３Ａを用いて説明した変換／暗号化ルーチン９８と同一なので、これ以上の説明は、以前した説明の繰り返しになる。ルーチン９８の説明を除いた分配コード化法１００の詳細な説明を以下に示す。

図３２について言及すれば、分配コード化法１４０は、初めに、ニブル毎に、入力データを判別することにより、最も頻繁に発生するニブル値を決定する（ステップ１４２）。次に、データは予め決められたニブルのセグメントに分離される（ステップ１４４）。セグメントの長さはある程度任意であるが、１０ニブルから数百ニブルの長さで配置されたセグメントは最も実行可能な傾向にある。初めのセグメントは検索され、その内容がニブル毎に、そのセグメントのいくつか

のニブルが最も頻繁に発生するニブルの値と合致するかどうかについて判別される（ステップ１４６）。

図３２、３３Ａについて言及すれば、データの流れは各々が１６ニブルを含む１００セグメントから構成されると仮定しよう。また、最も頻繁に発生するニブル値が１４（すなわち、２進数１１１０）であることを仮定しよう。各セグメントは最も頻繁に発生するニブルの発生に関して判別され、最も頻繁に発生するニブル値を含むセグメントに対して制御ビットCONT$を１に設定する（図３３Ａのセグメント２、３、１００参照）。セグメントが最も頻繁に発生するニブル値を含まなければ、CONT$値に０を設定する（図３３Ａのセグメント１、９９参照）。次に、最も頻繁に発生するニブル値を含む各セグメントに対して、その列の中のニブル値の配置を変数POS$に記録し、最も頻繁に発生するニブル値は各セグメントから取り除かれる。残った”穴”は、各セグメント内において右側のほとんどのニブルを”左方に移す”ことによって埋められる（ステップ１６０）。表６から、POS$値はアドレスされ、対応するMPV$値は記録される（図３３Ｂ）。次に、列は全MPV$の列のTOTMPV$を形成し、全CONT$の列TOTCONT$を形成する。最も頻繁に発生するニブル値が発生しないセグメントにおいて、全１６ニブル値は０から１４までの値の範囲でマップされる（ステップ１５２）。このマッピングは１５の基本値を用いる従来のパックルーチンの適用を可能とする（パックルーチンの例に関しては定義部を参照）。

表6

位置 (POS$)	マップされた位置値 (MPV$)	位置 (POS$)	マップされた位置値 (MPV$)
1	0000	9	0100
2	0010	10	0110
3	0001	11	0101
4	0011	12	0111
5	1000	13	1100
6	1010	14	1110
7	1001	15	1101
8	1011	16	1111

分配コード化法により実行された節約を以下に示す。

Ａ．　最も頻繁に発生するニブル値が発生しない列においては、そのセグメントに含まれる各１６ニブルに対して約１．４８８ビット節約できる。例えば、３２ニブル長のセグメントについて処理すれば、２．９７６ビット節約できる。

Ｂ．　最も頻繁に発生するニブル値が発生するセグメントにおいては、２つの違う節約に役立つ。第１には、最も頻繁に発生するニブルの位置より先に発生する全てのニブルに対して、これらを先に述べた方法でパックすることができる。この方法でパックすることにより、最も頻繁に発生するニブルより前に発生する全てのニブルに対して０．０９３ビット節約される。第２の節約の原因は、表６にマップされた位置値に示した方法にある。最も頻繁に発生するニブル値は１回以上発生するので、それらが発生する時、そのセグメントの上位半分の位置で発生するよりも下位半分の位置で発生する傾向がある。この平均を取ると、０にバイアスされたマップされた位置値（MPV$）を得られ、位置９から１６で見つけられる０よりも多くの０を位置１から８で得られる。実際この０のバイアスが発生すれば、それは、圧縮のための変換／暗号化ルーチン９８を通るように送られる

完全な候補となる列TOTMPV$に反映される。

Ｃ．　を列TOTCONT$が変換／暗号化ルーチン９８を通るように送ることにより、第３の節約の原因が示される。任意の均等に分配されたデータにおいて、最も頻繁に発生するニブル値は列の約６４．２％において発生し、列の約３５．８％において発生しない。このことは、変換／暗号化ルーチン９８を通る候補となるTOTCONT$のバイアスに反映される。

図３４について言及すれば、分配圧縮９２を用いてセグメント１から１００を処理した後、結果としての列は図３４に示される順序に基づいて連結される。出力列を元のファイルを再構成するために復元する時、復元処理は容易である。まず、列MOD(TOTCONT$)及び列MOD(TOTMPV$)は、変換／暗号化ルーチン９８により処理された復元情報を用いてすでに説明された方法で復元される。次に、TOTCONT$にアクセスすることにより、各パックされたブロックの状態を知る。ブロックが最も頻繁に発生するニブル値を含んでいなかったら、単純にその内容をアンパ

ックする（アンパック操作の例は定義部のパックルーチン参照）。セグメントが最も頻繁に発生するニブル値を含んでいたら、最も頻繁に発生するニブル値より先に発生するバイトをアンパックする。そこで、アンパックされた部分がブロックの残りの部分と連結される。分配圧縮９２はニブル圧縮が適用されたのと同一な方法でデータのブロックに連続的に用いられる。このように、ニブル圧縮９０及び分配圧縮９２は再帰的であり、それらが適用される時はいつでも、データ圧縮に対して効果的である。従って、少しだけの縮小が可能な時でさえ、その少しだけの縮小は再帰により拡大される。そこでは、”単一のパス”の圧縮技術を使用した時に可能な分よりも更に大きい縮小が成される。例えば、０．０９３ビットの縮小でさえも１００万回の再帰を実行すれば、９３．０キロビットの圧縮という結果になる。

図２、３Ｂについて言及すれば、この部分では、分配圧縮９２の処理とそれがどのようにしてデータを圧縮するためにパックルーチン及び変換／暗号化ルーチン９８を用いて動作するかを説明した。３つの圧縮ルーチンの３番目を次に説明する。

３．　直接ビット圧縮

図２、３Ａ、３Ｂ、３Ｃ、３５について言及すれば．直接ビット圧縮９４はブロック７０においてニブル圧縮９０及び分配圧縮９２のように動作する。従って、ソースファイル上で再帰的（反復的に）に使用できる。ニブル圧縮９０及び分配圧縮９２との相違点は、変換／暗号化ルーチン９８を使用しないことである。

直接ビット圧縮は圧縮するために単一のアルゴリズムを用いる。このアルゴリズムは図３５を用いて最も容易に説明される、直接ビットコード化アルゴリズム９５である。ソースファイルは読み込まれ、そのファイルから８ビットのワードが分析される。各特有のワードの値によって、予め決められた値がワード値から引かれ、バランス値を生成する。例えば、ケース１の範囲にあるワード値は０の制御ワードが割り当てられる。ケース２の範囲にあるワード値は１の制御ワードが割り当てられる。０のワード値（ケース９）は、１０００００１の制御ワードが割り当てられる。図３５に示すように、ケース２から８は、コード化ビット

数と制御ワードのビット数を加えると７になり、ケース１及び９は８となる。ケース２から９は分析ビットが割り当てられる。ケース１は分析ビットは必要としない。図３５の右側の欄は、各ケースについて、コード化ビットの数と制御ワードのビットの数と分析ビットの数とを各ケースに対して加えるとケース９を除いては８になり、ケース９では合計が９になる。

図３５、３６について言及すると、直接ビットコード化法９５を適用する一例において、入力列がBYTE1からBYTE5の５バイトから構成されると仮定しよう。BYTE1からBYTE5は、図３６に示される各値と同等の２進数である。３２のワード値を持つBYTE1はケース３の状態として配列され、そこでは、その３２という値から３２という値が引かれ、０のバランス値と、”１１”の制御値と、１の分析ビッ

トが生じる。同様にして、BYTE2、BYTE3、BYTE4に対するバランス値、制御値、分析ビットが図３５より得られる。次に、バランス出力列（BOUTS$）は、バランス値をEB1からEB5の順番に連結することにより、作成される。次に、出力列CW/ROUTS$は、EB1からEB5をその順番で連結し、その後にEB1からEB5の分析ビットを逆の順番で連結することより、作成される。

元の列STRING$を再構成するための（BOUTS$）と（CW/ROUTS$）のデコードは、図３７の復元木に従って行われる。CW/ROUTS$とBOUTS$の最後に残った列に対しては、デコードのためには分析ビットは不要である。従って、最後の分析ビット（図３６の例ではEB4$に示される）はCW/ROUTS$において元のワード値の再構成の可能性を失うことなく、除去可能である。よって、ファイルにおけるデータのエントロピーにかかわらず、直接ビットコード化法９５を用いてファイルを反復処理する時毎に、１ビットのセービングが保証されている。

直接ビットコード化に対する１つの方法が図３８Ａから３８Ｃに示される。初めに、入力列（STRING$）は、２つの出力列BOUTS$及びCW/ROUTS$を作成するために直接ビットコード化法により処理される（これは図３６に示す例の結果であり、図３８Ａに示される再帰レベル”０”の出力を形成する）。連続的な繰り返し処理において、列BOUTS$は、直接ビットコード化法により処理され、それにより

EB$(CW)とEB$(R)による列が連続的に作成される。この処理は残っている列BOUTS$が予め決められたサイズ（３２ビットのような）に縮小されるまで、続けられる。次に、図３８Ｂの４つの列が図のように構成され、図３８Ｃに示すように相互に連結される。更なる再帰が必要ならば、２番目の再帰カウンタは回数のトラックを保持し、３８Ｃでの列は３８Ａの方法論に戻される。従って、技術に熟練した者は、適当なカウンタを保持することにより、この発明の直接ビットコード化法は、入力列のビットパターンに関係なく、入力列を１ビット縮小する効果があることがわかる。種々のカウンタ及びレジスタを保有していくことは特定の”損失”が必要だけれども、十分な大きさのファイルに対して、このオーバヘッドは直接ビットコード化法により得られる節約と比較すると無意味である。

直接ビットコード化法において常に入力列から１ビット削り落とすという規則の例外は、入力バイトが０のワード値を持つ場合だけである。この例においては、ケース９の状態があり、そこでは、コード化ビットと制御ワードと分析ビットとが加えられ、９ビット長の列となる。この場合において、結合されたCW/ROUTS$及びBOTS$の長さが事実１ビット増加する。この望ましくない結果を取り除くために、均等に分配された１０２４バイトのデータにおいて、３ビットの増加の結果である４つの”０”ビットが平均して現れる（直接ビットコード化法の適用により得られた４ビットから、最後のバイト上の分析ビットを除去することにより節約された１ビットを引く）。しかし、作業領域をを１バイトから２バイトに変え、図３８に示す方法でコード化することにより、”０”ビットの値が発生する可能性は１／１６，７７７，２１６となる。更に４バイトの作業領域が選択されると、”０”ビットの値が発生する可能性は１／４，２９４，９６７，２９６となる。従って、十分に大きい作業領域を選択することにより、ワード値”０”が発生する可能性は非常に小さい。ワード値”０”が発生する場合においてさえ、その再帰での縮小は再帰の連続により更に拡大する。

この発明による圧縮法の好適実施例について説明してきたが、この発明の精神から離れることなく、この発明を説明するために選択された好適実施例に種々の変形又は追加を行ってもよい。例えば、圧縮法を説明するために、ここで選択さ

れた多数の種々のものが特定の長さを持って選択されている（１ビット、１ニブル、１ワード等のような）。多くの例においてこれらのワードのサイズは柔軟性があり、種々のアルゴリズムの処理に影響することなく、容易に変更できる。従って、保護を受けようとする主題は、添付のクレームにおいて定義されている内容を拡張し、それについての全ての等価物を含む。

【図１】

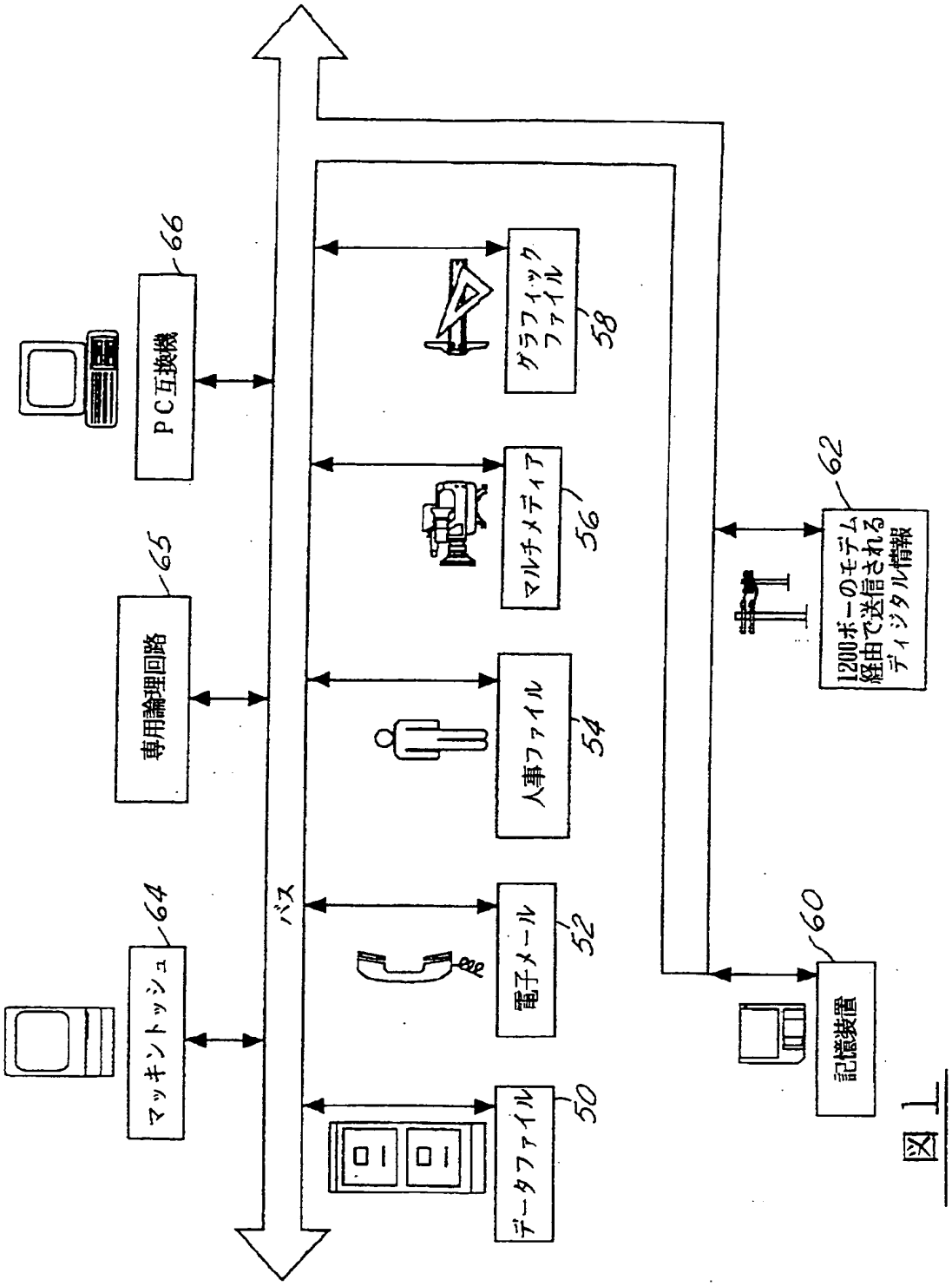

PC互換機
66
専用論理回路
65
マッキントッシュ
64
バス
データファイル
50
電子メール
52
人事ファイル
54
マルチメディア
56
グラフィックファイル
58
記憶装置
60
1200ボーのモデム経由で送信されるディジタル情報
62

図１

【図２】

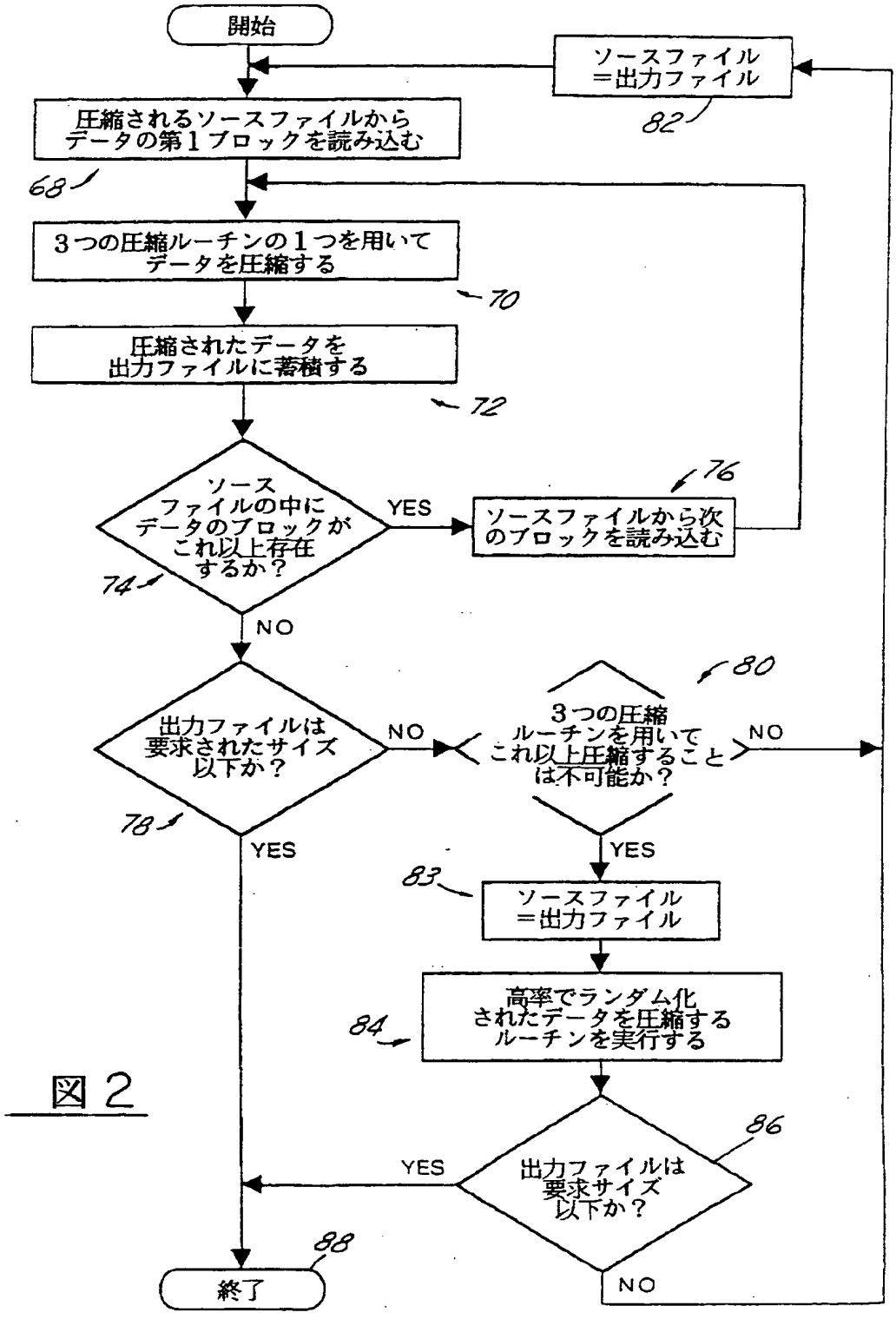
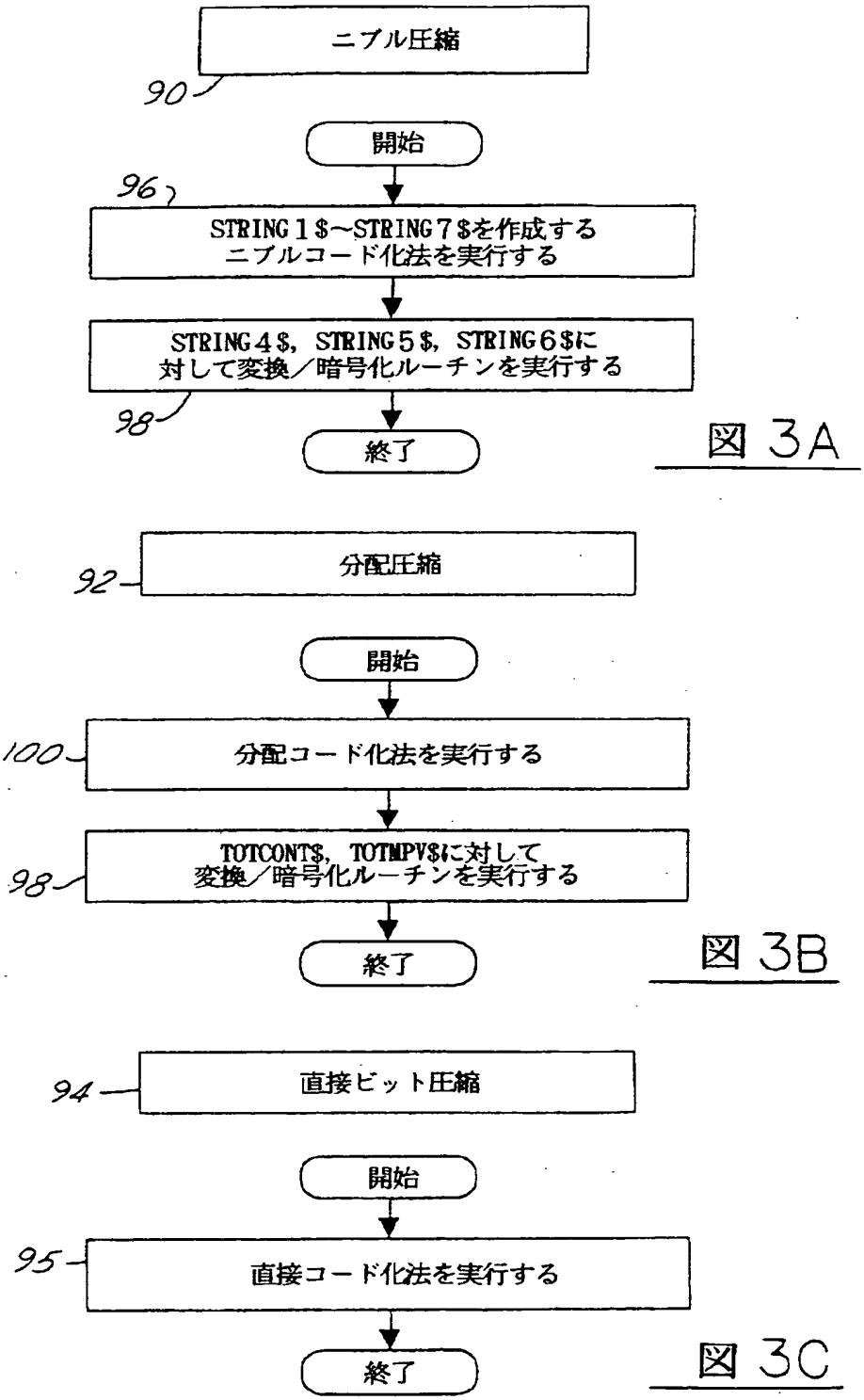

開始
圧縮されるソースファイルから
データの第１ブロックを読み込む
68
３つの圧縮ルーチンの１つを用いて
データを圧縮する
70
圧縮されたデータを
出力ファイルに蓄積する
72
ソース
ファイルの中に
データのブロックが
これ以上存在
するか？
74
YES
ソースファイルから次
のブロックを読み込む
76
NO
出力ファイルは
要求されたサイズ
以下か？
78
NO
３つの圧縮
ルーチンを用いて
これ以上圧縮すること
は不可能か？
80
NO
YES
YES
ソースファイル
＝出力ファイル
82
83
高率でランダム化
されたデータを圧縮する
ルーチンを実行する
84
出力ファイルは
要求サイズ
以下か？
86
YES
NO
終了
88

図２

【図３】


ニブル圧縮
90
開始
96
STRING１$～STRING７$を作成する
ニブルコード化法を実行する
STRING４$, STRING５$, STRING６$に
対して変換／暗号化ルーチンを実行する
98
終了
図３A
分配圧縮
92
開始
100
分配コード化法を実行する
TOTCONT$, TOTMPV$に対して
変換／暗号化ルーチンを実行する
98
終了
図３B
直接ビット圧縮
94
開始
95
直接コード化法を実行する
終了
図３C

【図４】

ニブルコード化法							
列 / 任意のニブルの値	STRING1$	STRING2$	STRING3$	STRING4$	STRING5$	STRING6$	STRING7$
0000	0	0	00	X	X	X	X
0001	0	0	10	X	X	X	X
0010	0	0	01	X	X	X	X
0011	0	0	11	X	X	X	X
0100	0	1	00	X	X	X	X
0101	0	1	10	X	X	X	X
0110	0	1	01	X	X	X	X
0111	0	1	11	X	X	X	X
1000	1	X	00	0	0	X	X
1001	1	X	10	0	0	X	X
1010	1	X	01	0	0	X	X
1011	1	X	11	0	0	X	X
1100	1	X	X	0	1	X	X
1101	1	X	X	1	X	1	X
1110	1	X	X	1	X	0	0
1111	1	X	X	1	X	0	1

X = 変化無し

図４

入力列 / 出力列	バイト1 ニブル1	ニブル2	バイト2 ニブル3	ニブル4	バイト3 ニブル5	ニブル6	バイト4 ニブル7	ニブル8	バイト5 ニブル9	ニブル10
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
STRING1$	0	0	0	0	0	0	0	0	1	1
STRING2$	0	0	0	0	1	1	1	1	X	X
STRING3$	00	10	01	11	00	10	01	11	00	10
STRING4$	X	X	X	X	X	X	X	X	0	0
STRING5$	X	X	X	X	X	X	X	X	0	0
STRING6$	X	X	X	X	X	X	X	X	X	X
STRING7$	X	X	X	X	X	X	X	X	X	X

X = 変化無し

図 5A

バイト6 ニブル11	ニブル12	バイト7 ニブル13	ニブル14	バイト8 ニブル15	ニブル16	総ビット数		0の総ビット数に対する比率
1010	1011	1100	1101	1110	1111		64	50 %
1	1	1	1	1	1	16		50.0 %
X	X	X	X	X	X	8		50.0 %
01	11	X	X	X	X	24		50.0 %
0	0	0	1	1	1	8		62.5 %
0	0	1	X	X	X	5		80.0 %
X	X	X	1	0	0	3		66.6 %
X	X	X	X	0	1	2		50.0 %
						66	64	

図 5B

【図６】

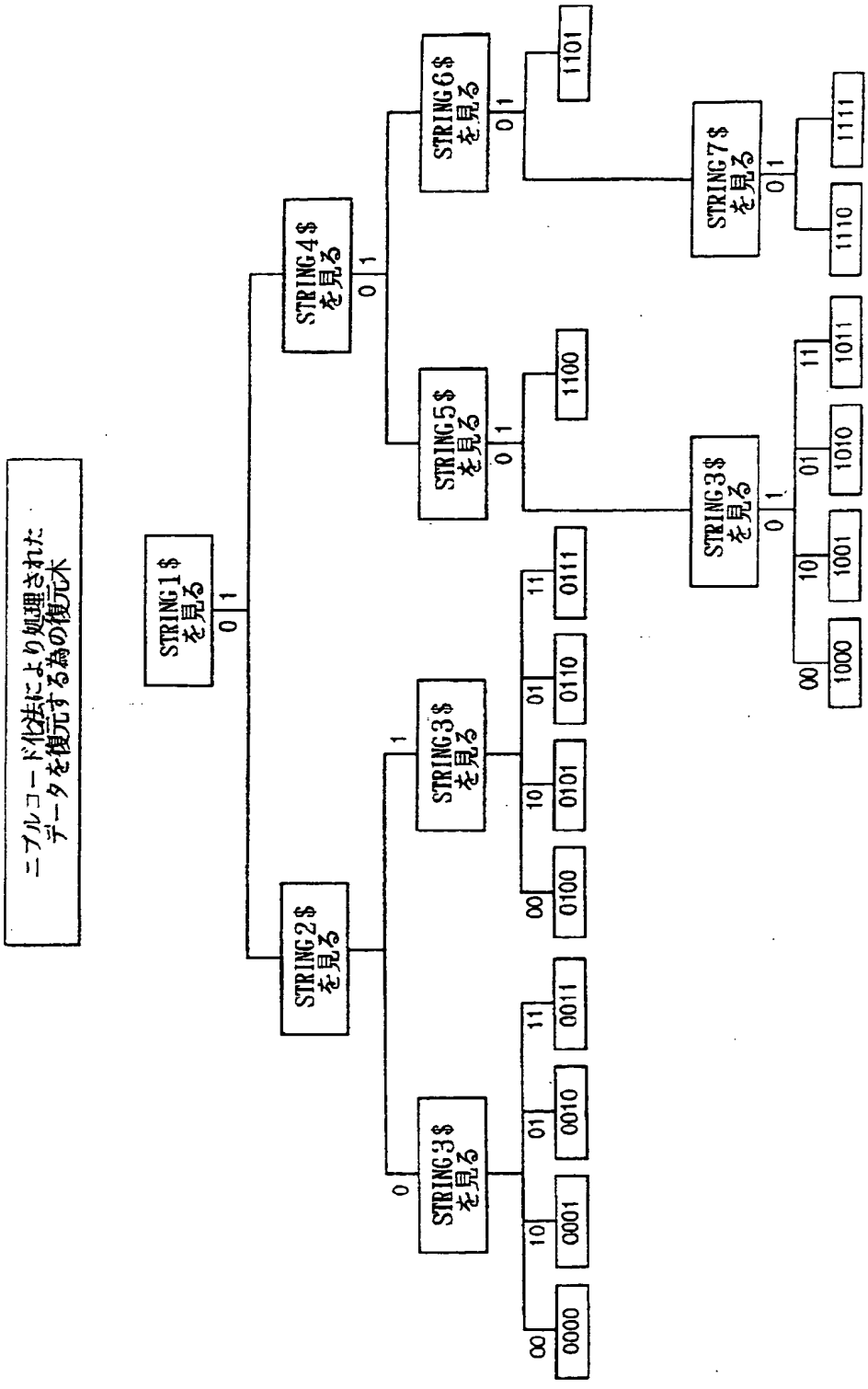

ニブルコード化法により処理された
データを復元する為の復元木
STRING1$
を見る
STRING2$
を見る
STRING3$
を見る
STRING4$
を見る
STRING5$
を見る
STRING6$
を見る
STRING7$
を見る
STRING3$
を見る
STRING3$
を見る
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

図６

【図７】

図４に示すニブルコード化法の使用によるコード化				
入力列 / 出力列	ニブル１ 0001	ニブル２ 0010	ニブル３ 0011	ニブル４ 1110
STRING1$	0	0	0	1
STRING2$	0	0	0	X
STRING3$	10	01	11	X
STRING4$	X	X	X	1
STRING5$	X	X	X	X
STRING6$	X	X	X	0
STRING7$	X	X	X	0
X＝変化無し				

図 7A

図６に示す復元木の使用によるニブル１の復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
STRING1$	0 0 0 1
STRING2$	0 0 0
STRING3$	1 0 0 1 1 1
STRING4$	1
STRING5$	
STRING6$	0
STRING7$	0
入力列＝0001	

図 7B

図６に示す復元木の使用によるニブル２の復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
STRING1$	0 0 1
STRING2$	0 0
STRING3$	0 1 1 1
STRING4$	1
STRING5$	
STRING6$	0
STRING7$	0
入力列＝00010010	

図 7C

【図７】

図６に示す復元木の使用によるニブル３の復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
STRING1$ STRING2$ STRING3$ STRING4$ STRING5$ STRING6$ STRING7$	0 1 0 1 1 1 0 0
入力列 = 000100100011	

図 7D

図６に示す復元木の使用によるニブル４の復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
STRING1$ STRING2$ STRING3$ STRING4$ STRING5$ STRING6$ STRING7$	1 1 0 0
入力列 = 0001001000111110	

図 7E

【図８】

ニブルコード化方法論に使用される
列のパッケージシーケンス

{ STRING1$ / STRING2$ / STRING3$ (最初の部分) /
STRING4$ / STRING5$ / STRING6$ / STRING7$ / STRING3$ (最後の部分)

図 8

【図９】

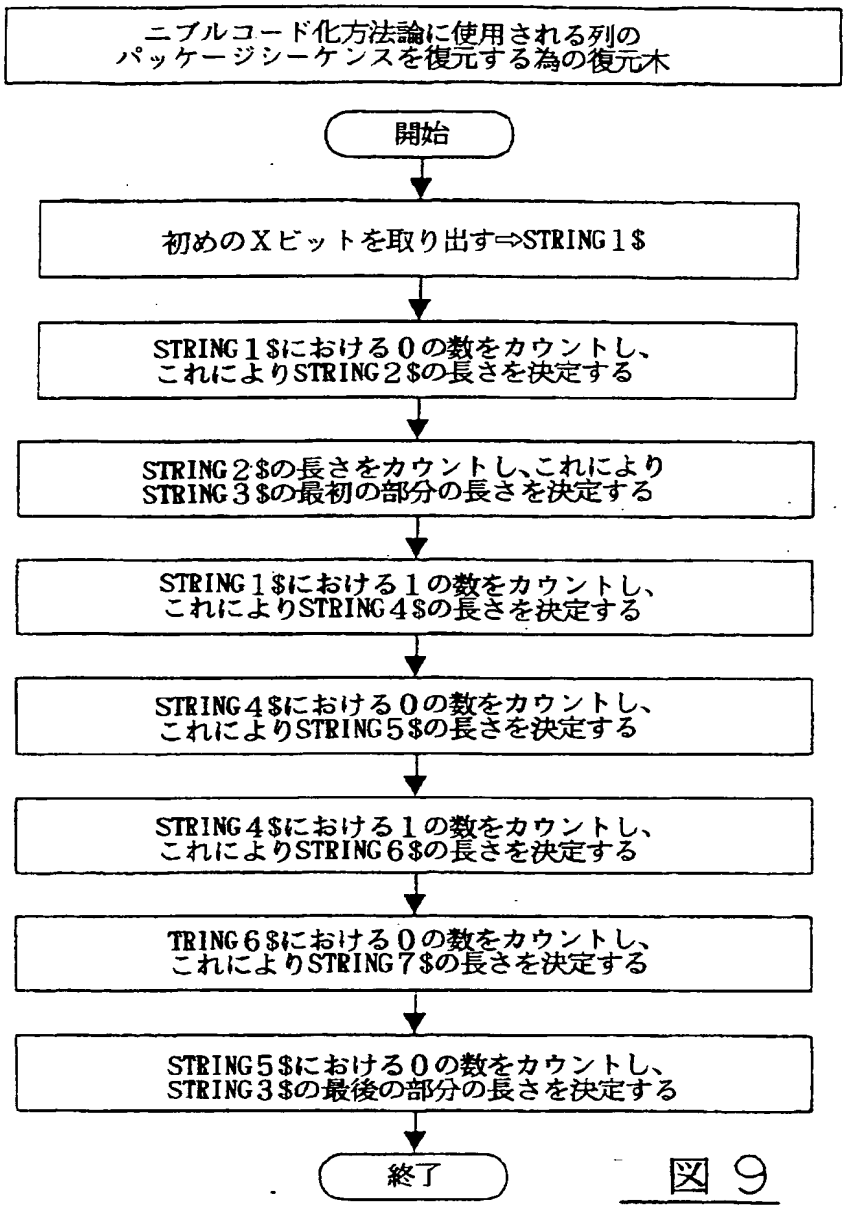

ニブルコード化方法論に使用される列の
パッケージシーケンスを復元する為の復元木
開始
初めのＸビットを取り出す⇨STRING１$
STRING１$における０の数をカウントし、
これによりSTRING２$の長さを決定する
STRING２$の長さをカウントし、これにより
STRING３$の最初の部分の長さを決定する
STRING１$における１の数をカウントし、
これによりSTRING４$の長さを決定する
STRING４$における０の数をカウントし、
これによりSTRING５$の長さを決定する
STRING４$における１の数をカウントし、
これによりSTRING６$の長さを決定する
TRING６$における０の数をカウントし、
これによりSTRING７$の長さを決定する
STRING５$における０の数をカウントし、
STRING３$の最後の部分の長さを決定する
終了

図９

【図１０】

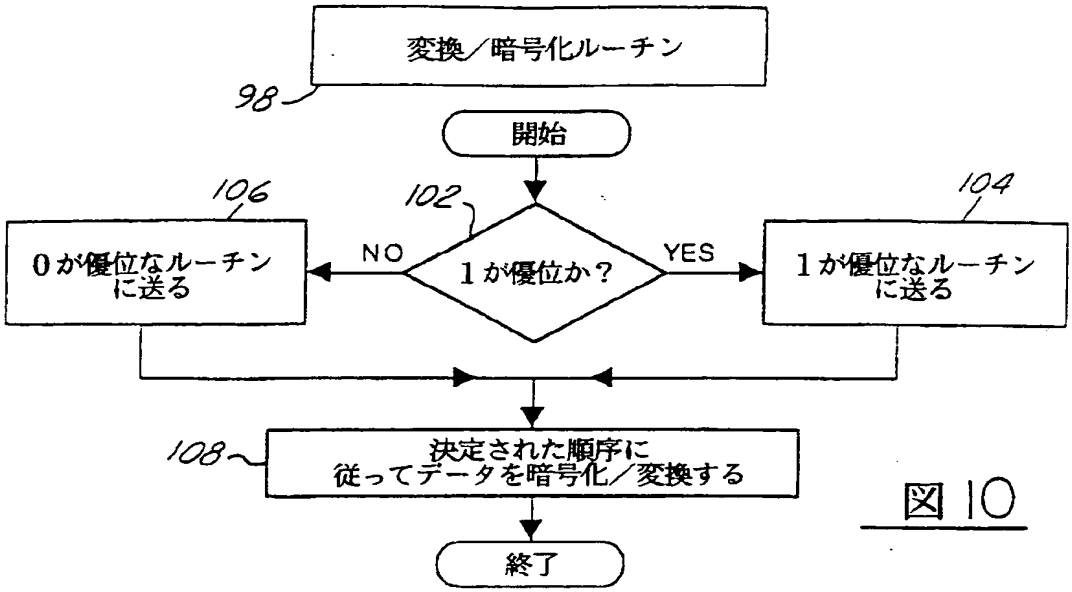


図 10

【図１１】

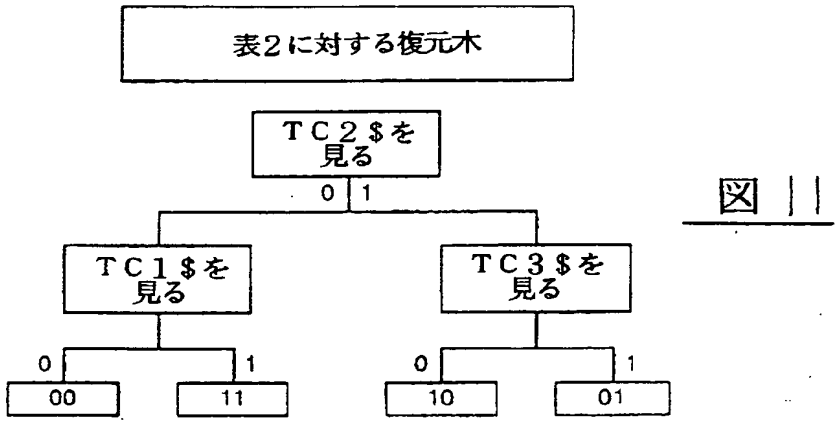


図 11

【図１２】

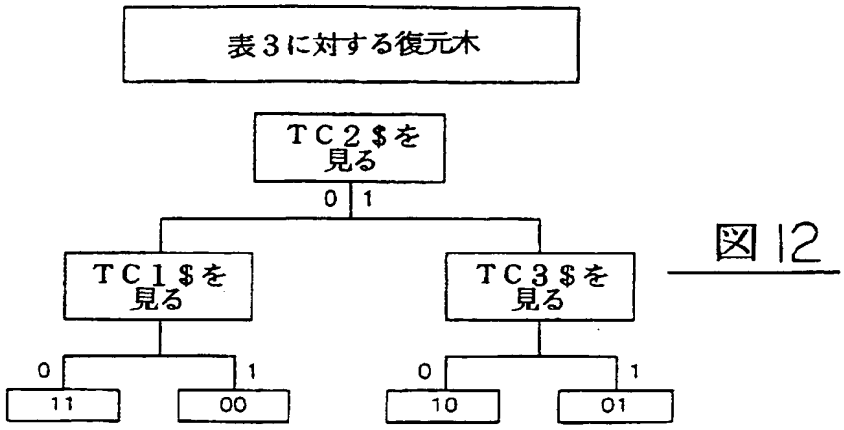


図12

【図１３】

表2に示す変換方法論の使用による変換										
出力列＼入力列	0	0	1	0	1	0	0	0	1	0
TC1$	0	X	X	0	X	0	X	0	0	X
TC2$	0	1	1	0	1	0	1	0	0	1
TC3$	X	0	0	X	0	X	0	X	X	0
X = 変化無し										

図 13A

図11に示す復元木の使用による復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
TC1$	0 0 0 0 0
TC2$	0 1 1 0 1 0 1 0 0 1
TC3$	0 0 0 0 0 0 0 0 0 0
入力列 = 00	

図 13B

図11に示す復元木の使用による復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9
TC1$	0 0 0 0
TC2$	1 1 0 1 0 1 0 0 1
TC3$	0 0 0 0 0
入力列 = 0010	

図 3C

図11に示す復元木の使用による復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
TC1$	0 0 0 0
TC2$	1 0 1 0 1 0 0 1
TC3$	0 0 0 0
入力列 = 001010	

図 13D

図11に示す復元木の使用による復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
TC1$	0 0 0 0
TC2$	0 1 0 1 0 0 1
TC3$	0 0 0
入力列 = 00101000	

図 13E

図11に示す復元木の使用による復元	
出力列	ビット配置 1 2 3 4 5 6 7 8 9 10
TC1$	0 0 0
TC2$	1 0 1 0 0 1
TC3$	0 0 0
入力列 = 0010100010	

図 13F

【図１４】

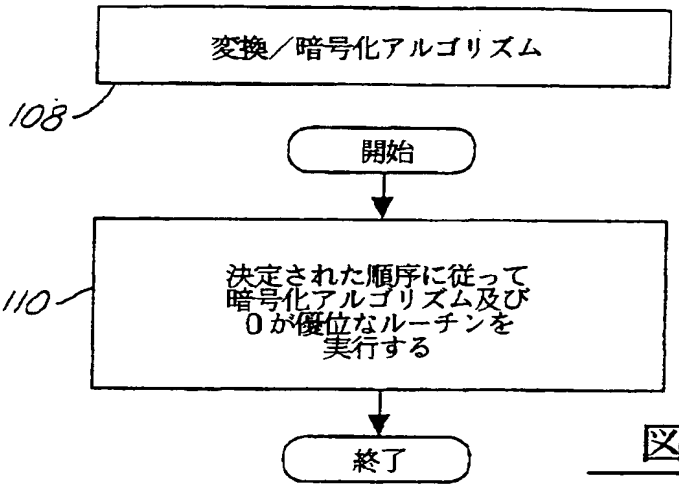


図14

【図１５】

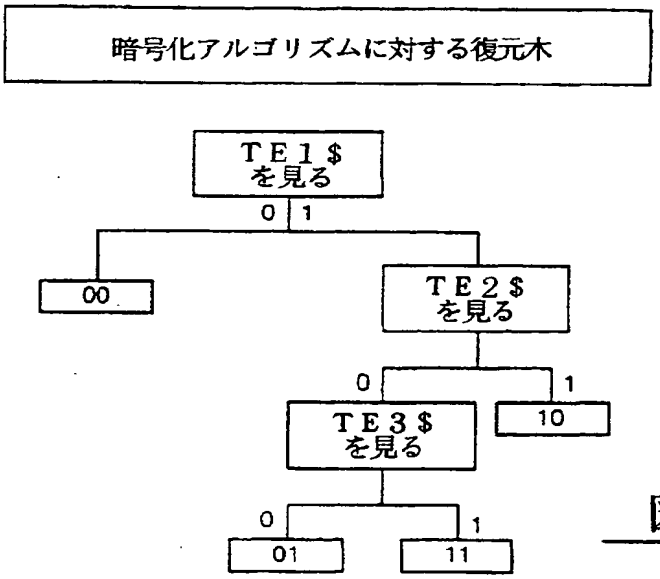


図15

【図１６】

表４に示す暗号化ルーチンの使用による暗号化										
出力列＼入力列	ビット配置									
	1	2	3	4	5	6	7	8	9	10
	0	0	1	0	0	0	1	0	0	0
TE1$	0	1	0	1	0					
TE2$	X	1	X	1	X					
TE3$	X	X	X	X	X					
X = 変化無し										

図16A

図15に示す復元木の使用による復元										
出力列	ビット配置									
	1	2	3	4	5	6	7	8	9	10
TE1$	◎	1	0	1	0					
TE2$	1	1								
TE3$										
入力列 = 00										

図16B

図15に示す復元木の使用による復元										
出力列	ビット配置									
	1	2	3	4	5	6	7	8	9	10
TE1$	1	0	1	0						
TE2$	1	1								
TE3$										
入力列 = 0010										

図16C

図15に示す復元木の使用による復元										
出力列	ビット配置									
	1	2	3	4	5	6	7	8	9	10
TE1$	◎	1	0							
TE2$	1									
TE3$										
入力列 = 001000										

図16D

図15に示す復元木の使用による復元										
出力列	ビット配置									
	1	2	3	4	5	6	7	8	9	10
TE1$	1	0								
TE2$	1									
TE3$										
入力列 = 00100010										

図16E

図15に示す復元木の使用による復元										
出力列	ビット配置									
	1	2	3	4	5	6	7	8	9	10
TE1$	◎									
TE2$										
TE3$										
入力列 = 0010001000										

図16F

0が優位なルーチン

入力列 STRING4$ / 出力列	4	8	12	16	20	24	28	32	36	40
	0010	1000	1000	1000	0010	0000	0001	1010	0000	0010
TC1$	0 X X 0	X 0	X 0	0 X	X X	0 0	0 X	X X	0 0	0 X
TC2$	0 1 1 0	1 0	1 0	0 1	0 0	0 1	1 1	0 0	0 1	
TC3$	X 0 0 X	0 X	0 X	X 0	X X	X 1	0 0	X X	X 0	

64ビット－64ビット＝変換法により除去されるのは0ビット

X = 変化無し

図17A

44	48	52	56	60	64	列における総ビット数	0の総ビット数に対する比率（すなわち0のバイアス）
0000	0000	1000	0011	0000	0101		
0 0 0 0	X 0	0 1	0 0	X X		← 64ビット	50/64 ⇒ .78125
0 0 0 0	1 0	0 0	0 0	1 1		← 20ビット	19/20 ⇒ .95000
X X X X	0 X	X X	X X	1 1		← 32ビット	20/32 ⇒ .62500
						← 12ビット	9/12 ⇒ .75000
						64ビット	

図17B

【図１８】

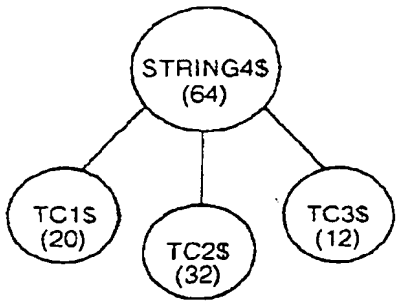
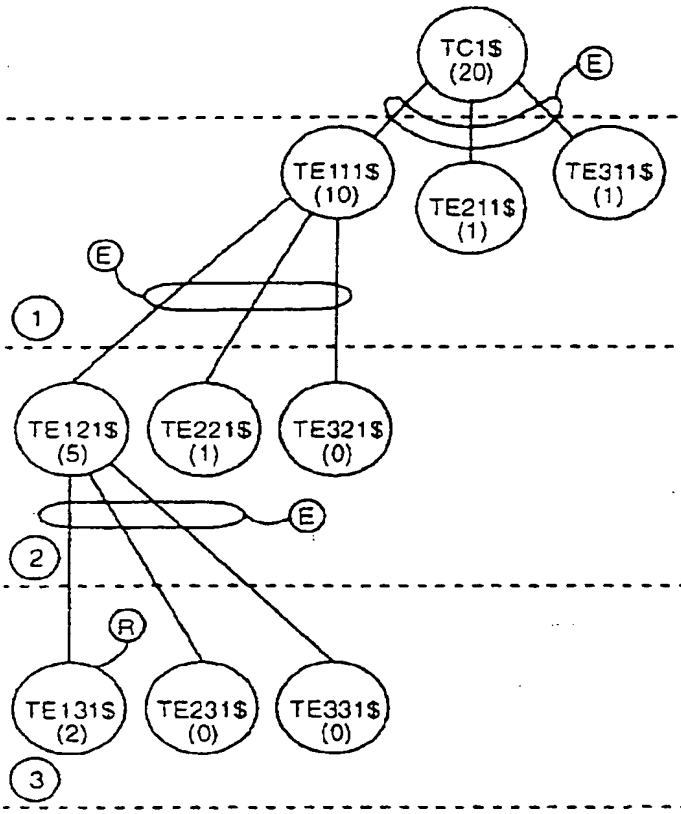


図 18

【図１９】



列の長さ

TC1$	TE111$	TE211$	TE311$
20	-	-	-
20	12	1	1
20	8	1	1
20	6	1	1

ＴＣ１＄の圧縮された長さ＝ 6+1+1
＝ 8 ビット

総縮小 = 20-8
＝ 12 ビット

縮小率 = 12/20
＝ 60%

図 19

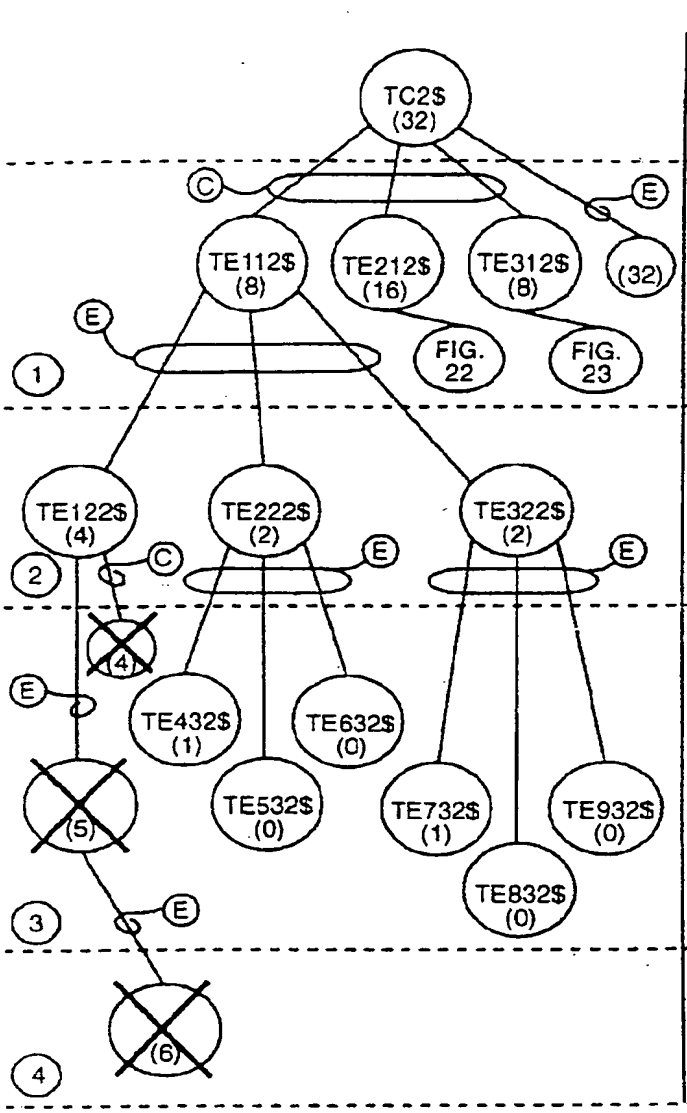
【図20】

TC1$に暗号化サブルーチンを使用する再帰法

暗号化する列（TC1$）／暗号化サブルーチンによる結果（テーブル4）	4 8 12 16 20		0の総ビット数に対する比率（すなわち0のバイアス）
	00000000000000000100	← 20ビット	19/20 ⇒ .95000
TE111$	0 0 0 0 0 0 0 0 1 0	← 10ビット	9/10 ⇒ 0.90000
TE211$	X X X X X X X X 0 X	← 1ビット	1/1 ⇒ 1.00000
TE311$	X X X X X X X X 0 X	← +1ビット	1/1 ⇒ 1.0000
X＝変化無し	20ビット−12ビット＝暗号化サブルーチンを通る第1のパスにおいて除去された8ビット	12ビット	
TE111$を暗号化サブルーチンを通るように送る（すなわちTE111$を再帰する）／TE111$／暗号化サブルーチンによる結果	4 8 12 16 20		
	0000000010	← 10ビット	9/10 ⇒ .95000
TE121$	0 0 0 0 1	← 5ビット	4/5 ⇒ 0.80000
TE221$	X X X X 1	← 1ビット	0/1 ⇒ 0.0000
TE321$	X X X X X	← +0ビット	N/A
X＝変化無し	10ビット−6ビット＝暗号化サブルーチンを通る第2のパスにおいて除去された4ビット	6ビット	
TE121$を暗号化サブルーチンを通るように送る（すなわちTE121$を再帰する）／TE121$／暗号化サブルーチンによる結果	4 8 12 16 20		
	0 0 0 0 1	← 5ビット	4/5 ⇒ 0.80000
TE131$	0 0	← 2ビット	2/2 ⇒ 1.00000
TE231$	X X	← 0ビット	N/A
TE331$	X X	← 0ビット	N/A
X＝変化無し	残余＝"1" 5ビット−3ビット＝暗号化サブルーチンを通る第3のパスにおいて除去された2ビット	+1ビット 3ビット	節約できるビット数の合計＝8＋4＋2＝14ビット

図20

【図２１】



列の長さ

TC2$	TE112$	TE212$	TE312$
32	-	-	-
32	8	16	8
32	8	15	8
32	6	13	7
32	6	12	7

最終的なＴＣ２＄の長さ = 6+12+7
= 25 ビット

総縮小= 32-25
= 7 ビット

縮小率 = 7/32
= 21.875%

図 21

【図２２】

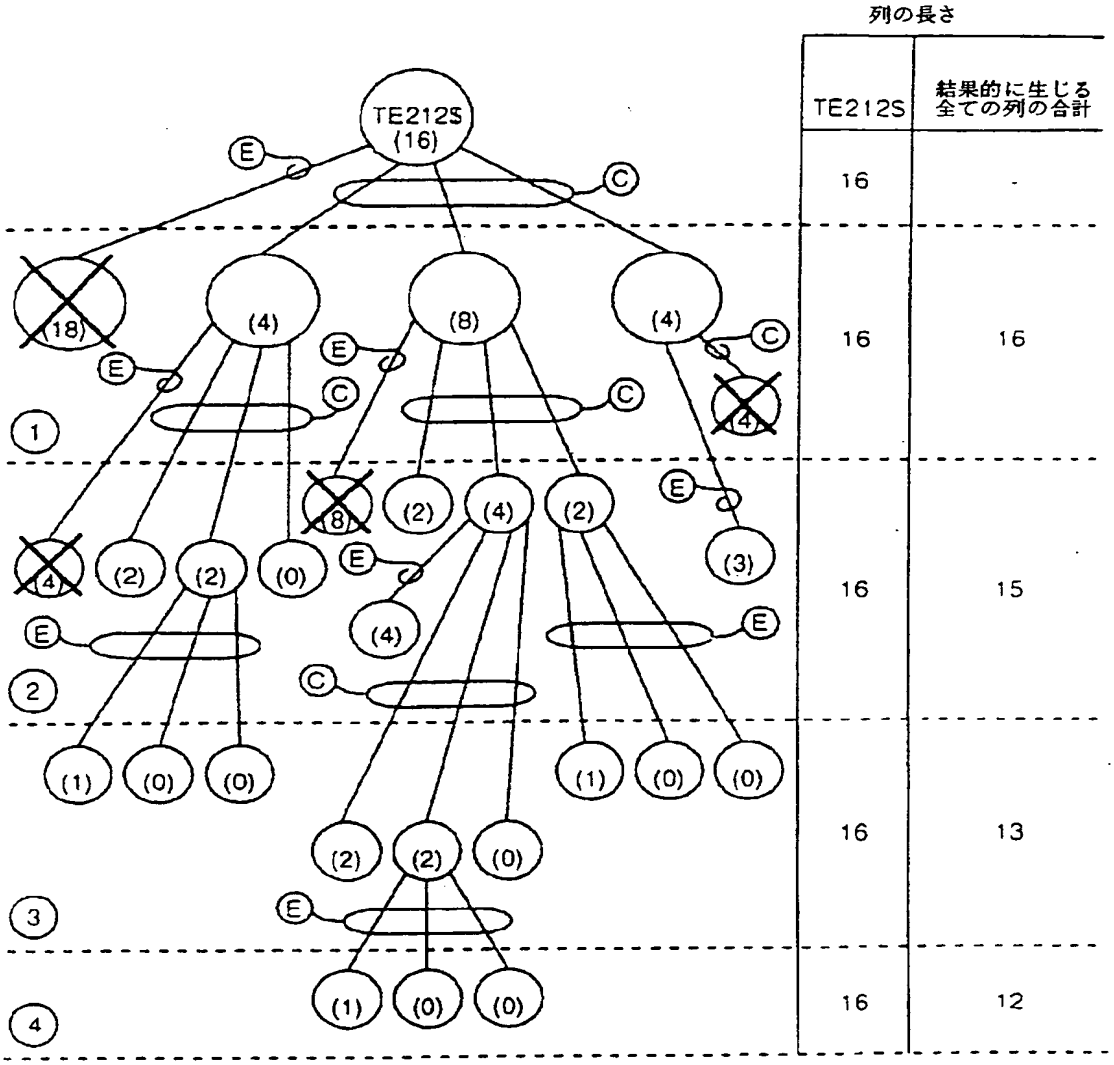
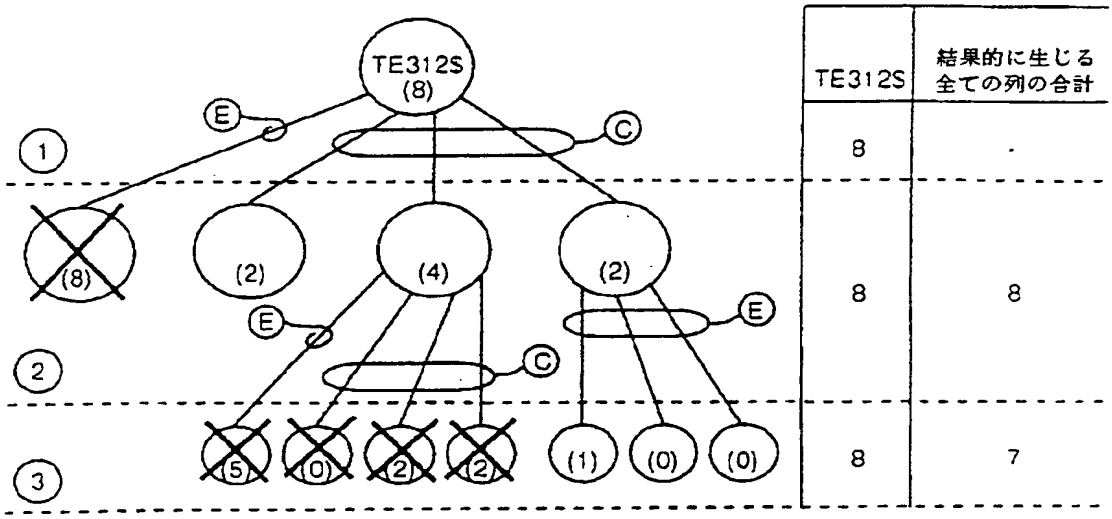

列の長さ
TE212S
結果的に生じる全ての列の合計
TE212S (16)
16
-
(18)
(4)
(8)
(4)
16
16
1
(8)
(2)
(4)
(2)
(3)
(4)
(2)
(2)
(0)
(4)
16
15
2
(1)
(0)
(0)
(1)
(0)
(0)
(2)
(2)
(0)
16
13
3
(1)
(0)
(0)
16
12
4

図 22

【図２３】



	列の長さ	
	TE312S	結果的に生じる全ての列の合計
1	8	
2	8	8
3	8	7

図 23

【図２４】

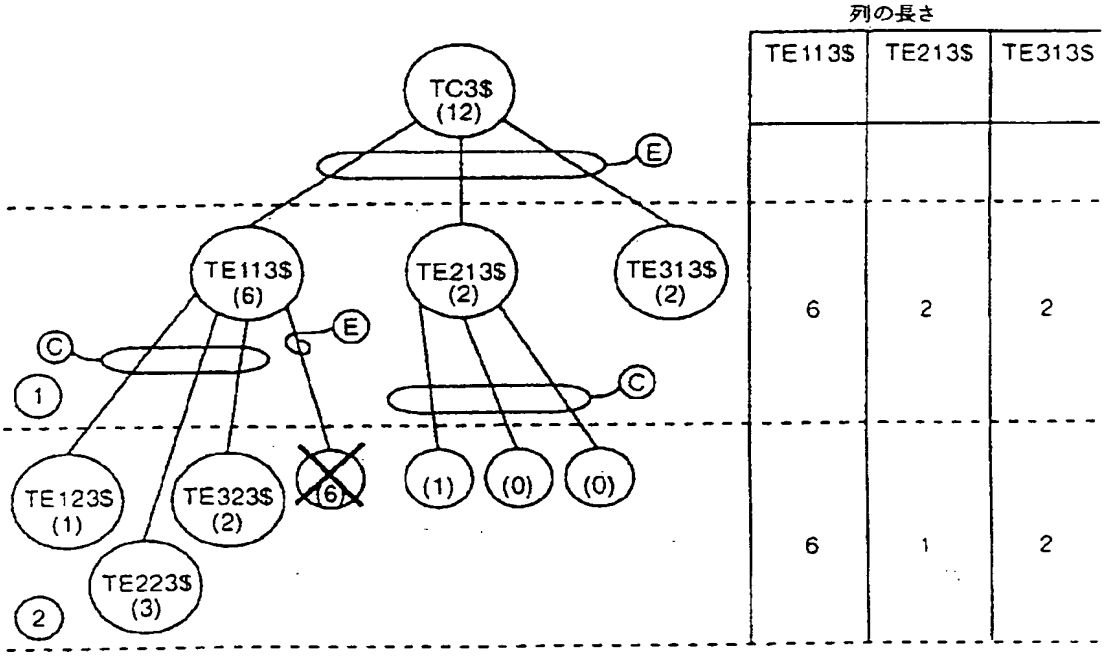
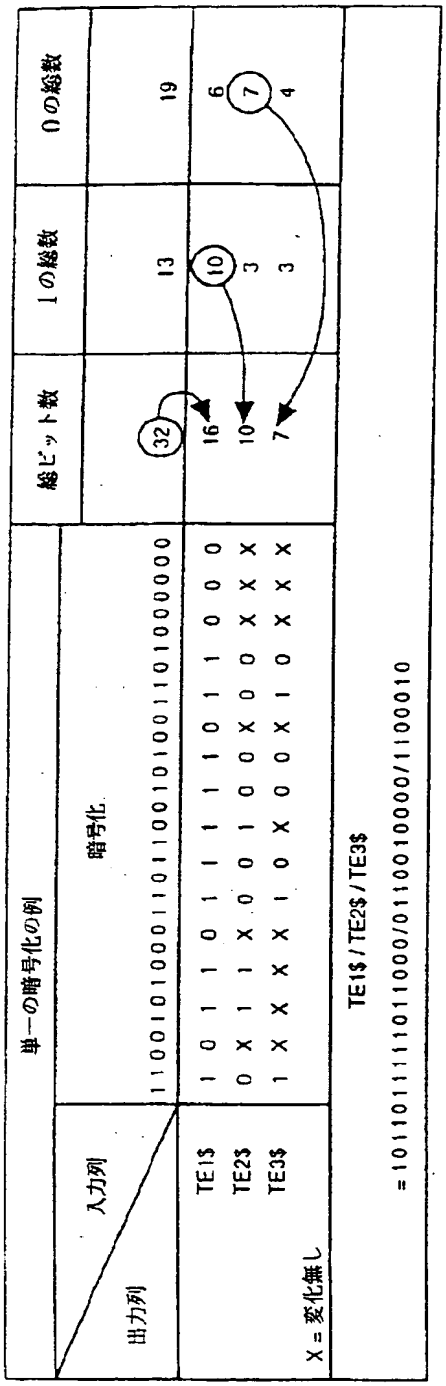

列の長さ
TE113$
TE213$
TE313$
TC3$
(12)
E
TE113$
(6)
TE213$
(2)
TE313$
(2)
C
E
C
1
TE123$
(1)
TE323$
(2)
(6)
(1)
(0)
(0)
TE223$
(3)
2
6
2
2
6
1
2
最終的なＴＣ３＄の長さ= 6+1+2
= 9ビット
総縮小 = 12-9
= 3ビット
縮小率= 3/12
= 25%

図 24

【図２５】


単一の暗号化の例
入力列
出力列
暗号化
1100101010001101100101001101000000
総ビット数
1の総数
0の総数
TE1$
TE2$
TE3$
X＝変化無し
1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 0 0 0
0 X 1 1 X 0 0 1 0 0 X 0 0 X X X X
1 X X X X 1 0 X 0 0 X 1 0 X X X X
32
16
10
7
19
13
3
3
6
4
TE1$ / TE2$ / TE3$
=10110111111011000/01100100000/1100010

【図２６】

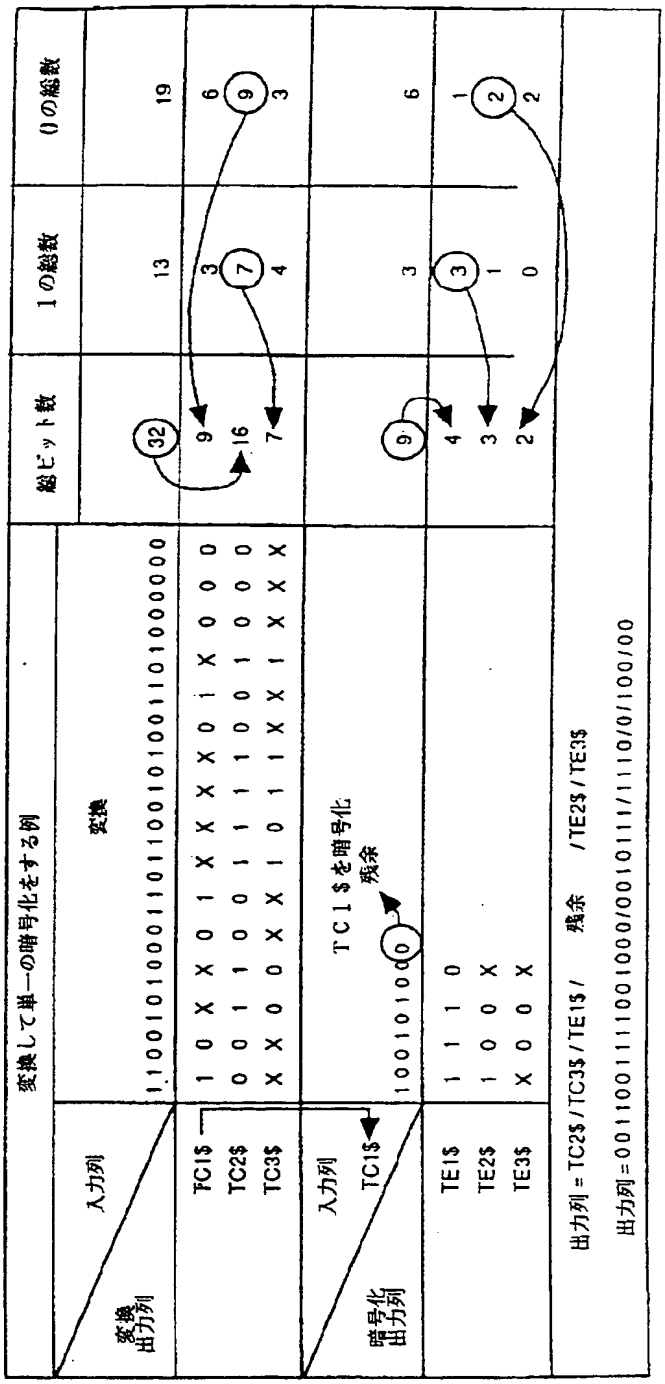

変換して単一の暗号化をする例
入力列
変換出力列
変換
1100101000011011001010011011010000000
TC1$
TC2$
TC3$
1 0 X X 0 1 X X X X 0 1 X 0 0 0 0
0 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 0
X X 0 0 X X 1 0 1 1 X X 1 X X X X
総ビット数
32
9
16
7
1の総数
13
3
7
4
0の総数
19
6
9
3
入力列
TC1$
暗号化出力列
TC1$を暗号化
残余
1 0 0 1 0 1 0 0 0
9
4
3
2
3
1
0
6
1
2
2
TE1$
TE2$
TE3$
1 1 1 1 0
1 0 0 X
X 0 0 X
出力列 = TC2$ / TC3$ / TE1$ / 残余 / TE2$ / TE3$
出力列 = 0011001111100100001001011111/1110/0/100/00

図26

【図２７】

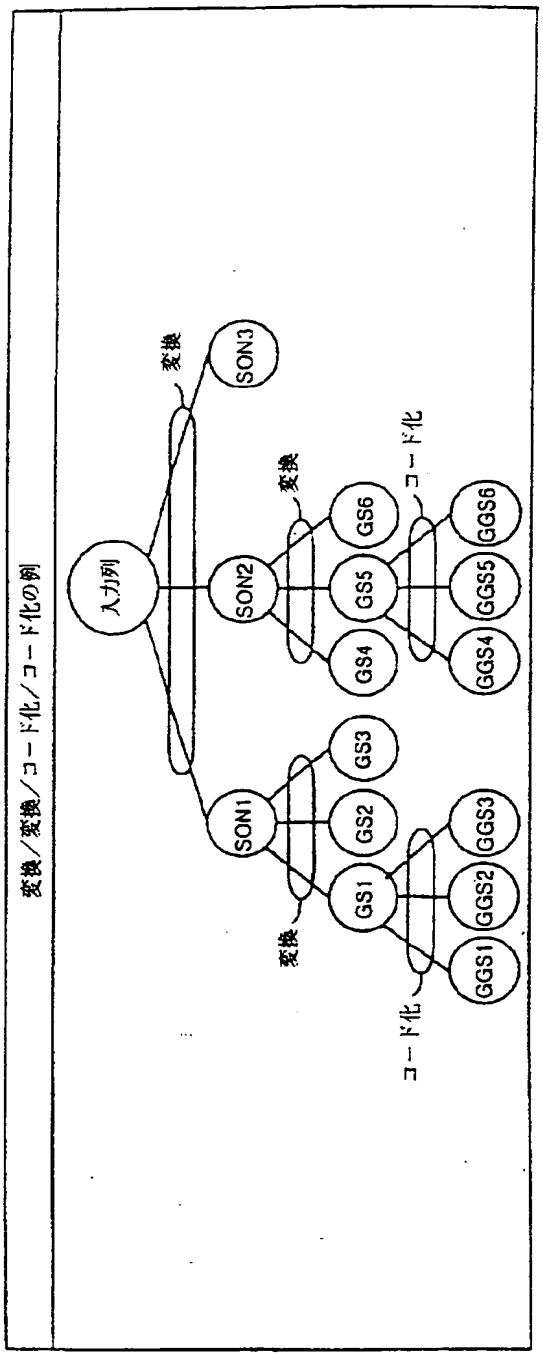


図27A

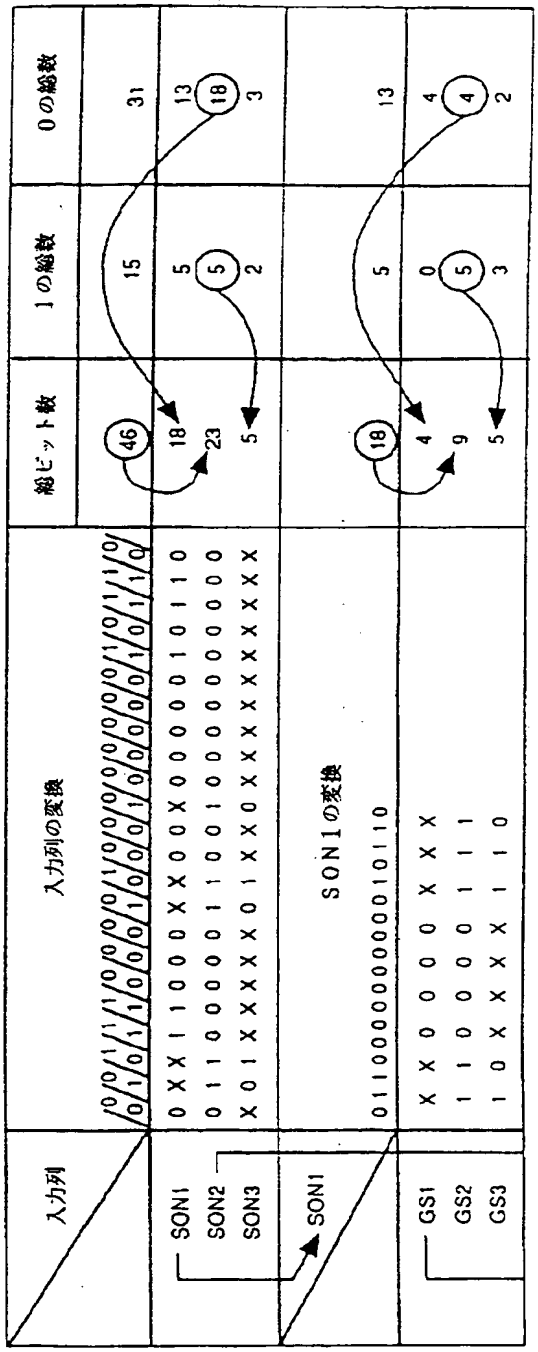


図27B

【図２７】

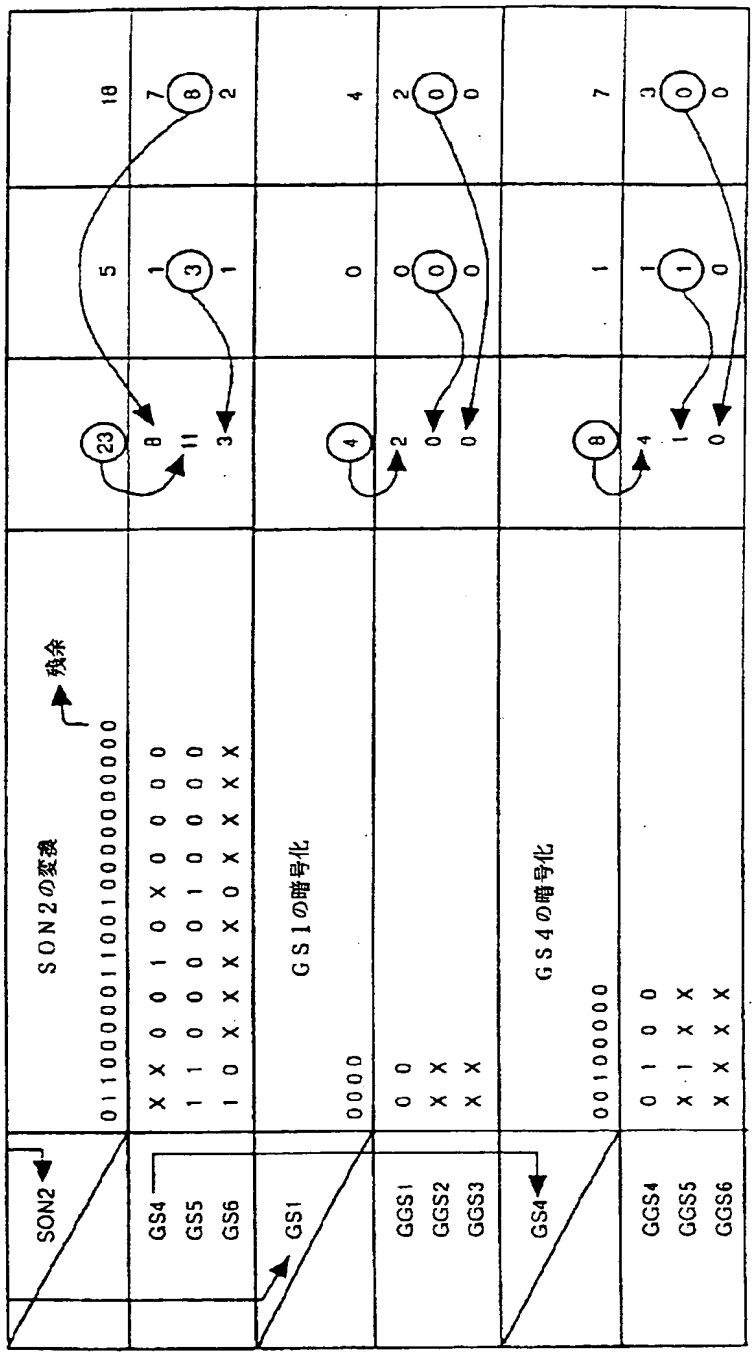

SON2
SON2の変換
残余
01100000110010000000000
GS4
GS5
GS6
X X 0 0 1 0 X 0 0 0 0
1 1 0 0 0 0 1 0 0 0 0
1 0 X X X X 0 X X X X
GS1
GS1の暗号化
0000
GGS1
GGS2
GGS3
0 0
X X
X X
GS4
GS4の暗号化
00100000
GGS4
GGS5
GGS6
0 1 0 0
X 1 X X
X X X X
18
7
8
2
5
1
3
1
23
8
11
3
4
0
0
0
4
2
0
0
7
1
1
0
8
4
3
0
1
0

図27C

【図２８】

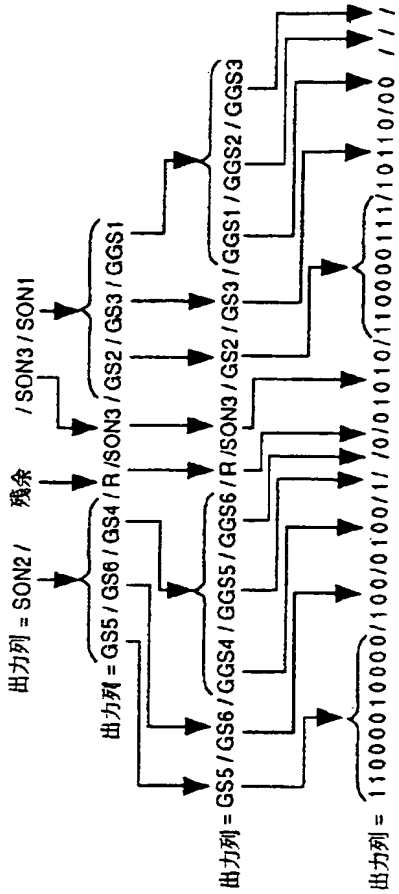

出力列 = SON2 / SON3 / SON1
出力列 = GS5 / GS6 / GS4 / GS5 / GS6 / R / 残余 / R / SON3 / GS2 / GS3 / GGS1
出力列 = GS5 / GS6 / GGS4 / GGS5 / GGS6 / R / SON3 / GS2 / GS3 / GGS1 / GGS2 / GGS3
出力列 = 1100001000/100/0100/1/ /0/01010/1100001111/10110/00 / / /

図28

【図２９】

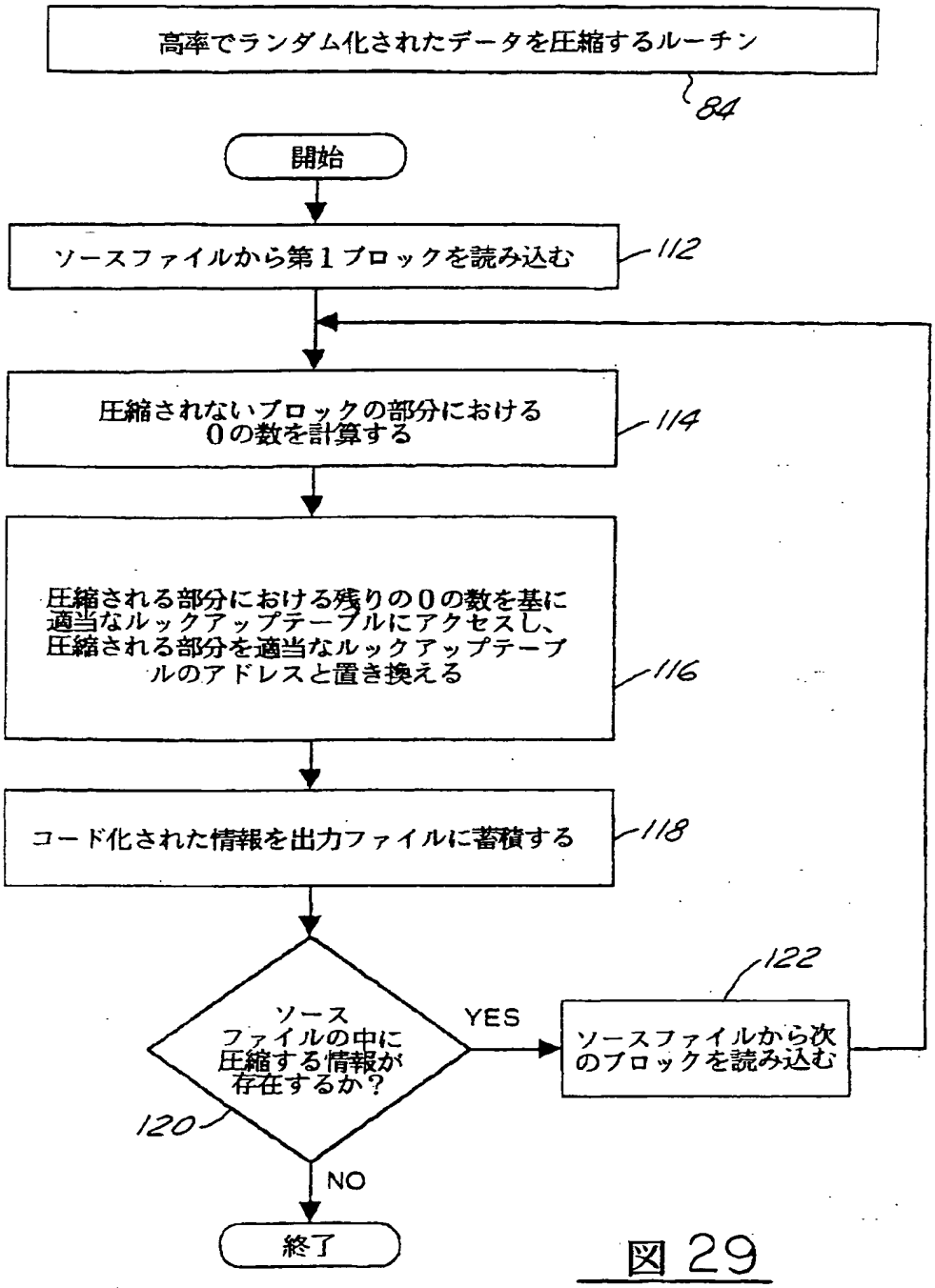

高率でランダム化されたデータを圧縮するルーチン
84
開始
ソースファイルから第１ブロックを読み込む
112
圧縮されないブロックの部分における０の数を計算する
114
圧縮される部分における残りの０の数を基に適当なルックアップテーブルにアクセスし、圧縮される部分を適当なルックアップテーブルのアドレスと置き換える
116
コード化された情報を出力ファイルに蓄積する
118
ソースファイルの中に圧縮する情報が存在するか？
120
YES
122
ソースファイルから次のブロックを読み込む
NO
終了

図 29

【図３０】

	ルックアップテーブル#1	0が9ケのビットパターン
1	0000	0000000001
2	0001	0000000010
3	0010	0000000100
4	0011	0000001000
5	0100	0000010000
8	1000	0010000000
9	1001	0100000000
10	1010	1000000000

図 30A

	ルックアップテーブル#2	0が8ケのビットパターン
1	000000	0000000011
2	000001	0000000101
3	000010	0000000110
4	000011	0000001010
5	000100	0000001100
43	101011	0110000000
44	101100	1010000000
45	101101	1100000000

図 30B

【図３１】

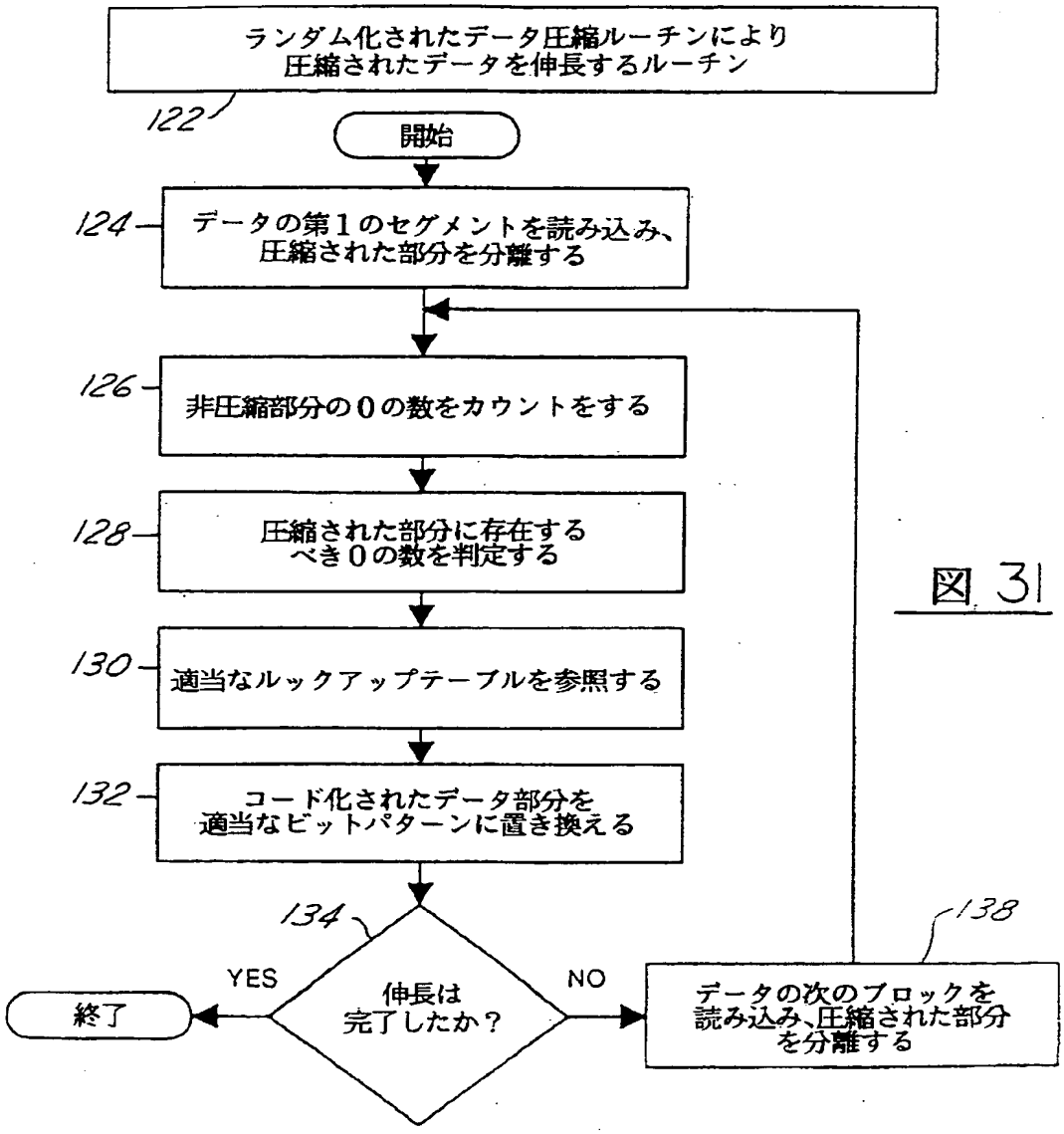

ランダム化されたデータ圧縮ルーチンにより
圧縮されたデータを伸長するルーチン
122
開始
124
データの第１のセグメントを読み込み、
圧縮された部分を分離する
126
非圧縮部分の０の数をカウントをする
128
圧縮された部分に存在する
べき０の数を判定する
130
適当なルックアップテーブルを参照する
132
コード化されたデータ部分を
適当なビットパターンに置き換える
134
伸長は
完了したか？
YES
終了
NO
138
データの次のブロックを
読み込み、圧縮された部分
を分離する

図 31

【図３２】

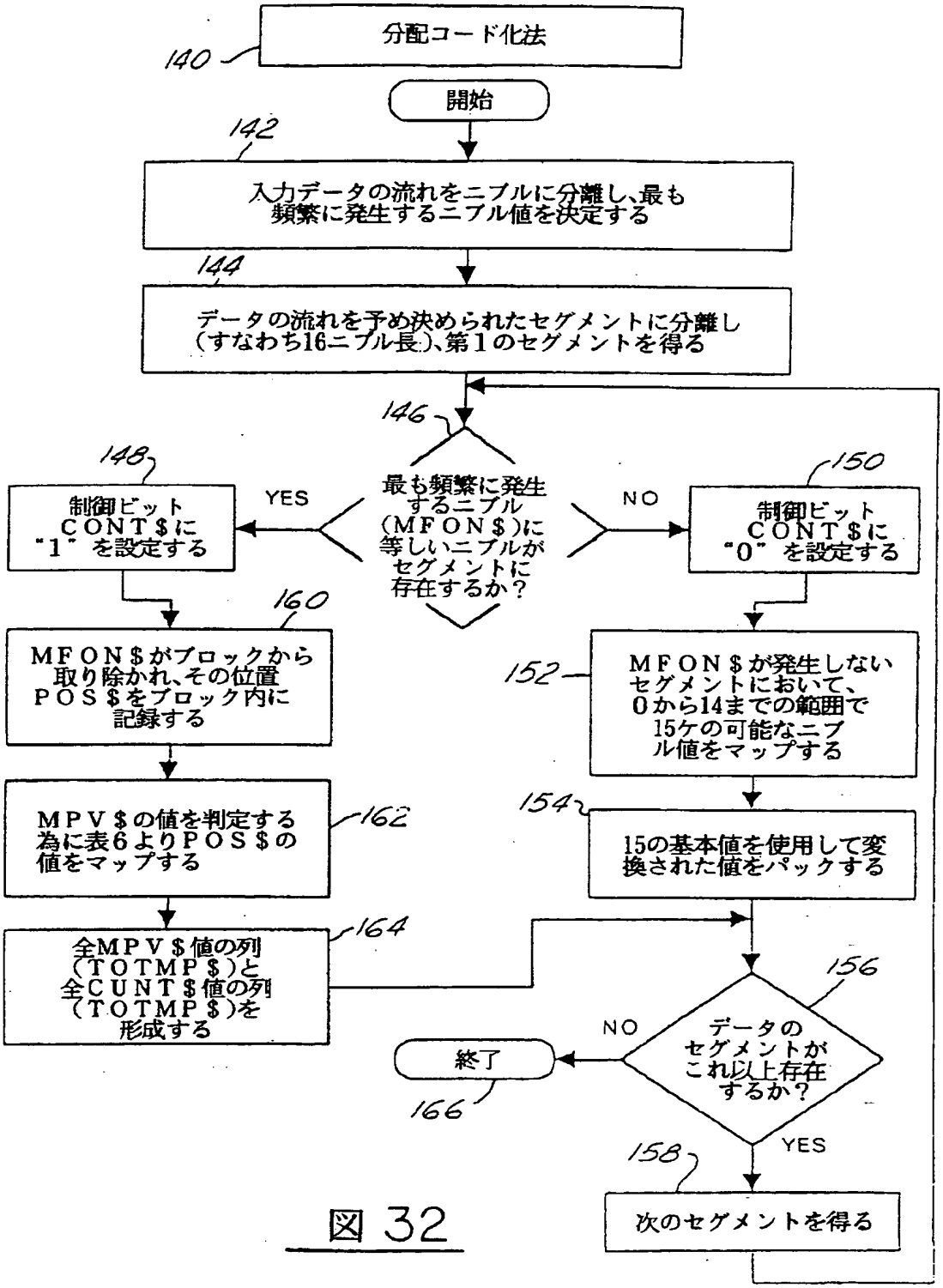
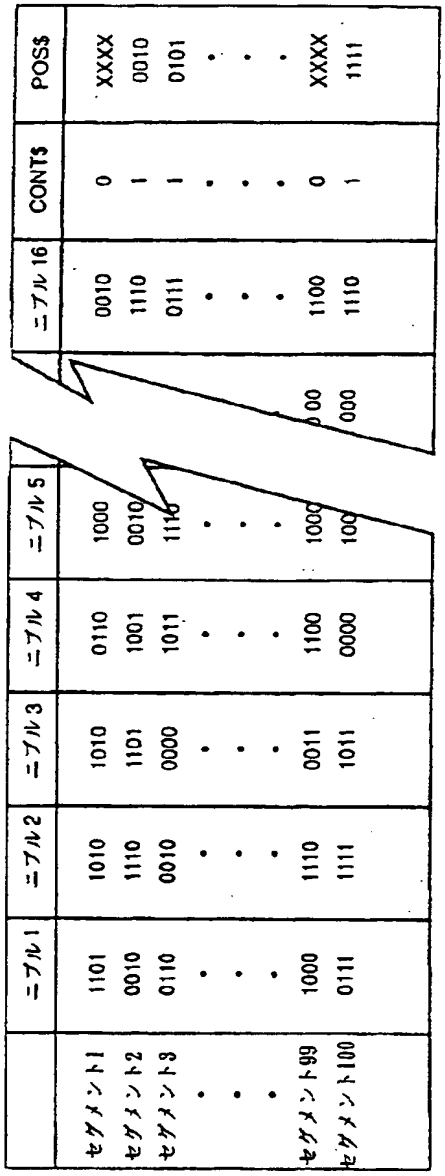

分配コード化法
140
開始
142
入力データの流れをニブルに分離し、最も頻繁に発生するニブル値を決定する
144
データの流れを予め決められたセグメントに分離し（すなわち16ニブル長）、第１のセグメントを得る
146
最も頻繁に発生するニブル（ＭＦＯＮ＄）に等しいニブルがセグメントに存在するか？
YES
NO
148
制御ビット ＣＯＮＴ＄に "1" を設定する
150
制御ビット ＣＯＮＴ＄に "0" を設定する
160
ＭＦＯＮ＄がブロックから取り除かれ、その位置ＰＯＳ＄をブロック内に記録する
152
ＭＦＯＮ＄が発生しないセグメントにおいて、０から14までの範囲で15ケの可能なニブル値をマップする
162
ＭＰＶ＄の値を判定する為に表６よりＰＯＳ＄の値をマップする
154
15の基本値を使用して変換された値をパックする
164
全ＭＰＶ＄値の列（ＴＯＴＭＰ＄）と全ＣＵＮＴ＄値の列（ＴＯＴＭＰ＄）を形成する
156
データのセグメントがこれ以上存在するか？
NO
YES
終了
166
158
次のセグメントを得る

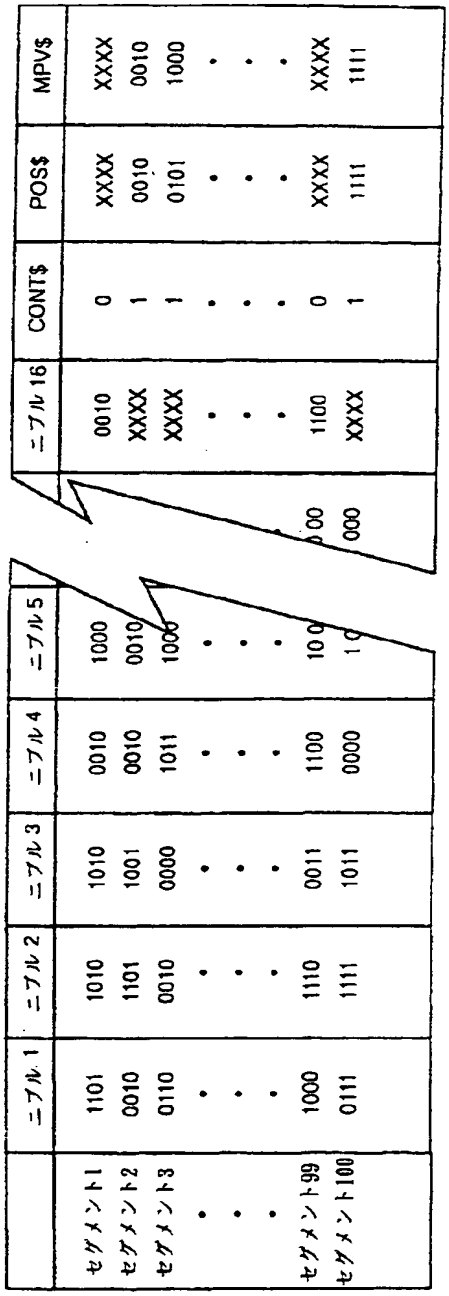
図 32

【図３３】



	ニブル1	ニブル2	ニブル3	ニブル4	ニブル5	ニブル16	CONT$	POS$
セグメント1	1101	1010	1010	0110	1000	0010	0	XXXX
セグメント2	0010	1110	1101	1001	0010	1110	1	0010
セグメント3	0110	0010	0000	1011	1110	0111	1	0101
・	・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・	・
セグメント99	1000	1110	0011	1100	1000	1100	0	XXXX
セグメント100	0111	1111	1011	0000	100	1110	1	1111

図 33A



	ニブル1	ニブル2	ニブル3	ニブル4	ニブル5	ニブル16	CONT$	POS$	MPV$
セグメント1	1101	1010	1010	0010	1000	0010	0	XXXX	XXXX
セグメント2	0010	1101	1001	0010	0010	XXXX	1	0010	0010
セグメント3	0110	0010	0000	1011	1000	XXXX	1	0101	1000
・	・	・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・	・	・
セグメント99	1000	1110	0011	1100	10 0	1100	0	XXXX	XXXX
セグメント100	0111	1111	1011	0000	1 0	XXXX	1	1111	1111

図 33B

【図３４】

図34

出力例＝MOD(TOTCONT$)/MOD(TOTMVP$)/パックされたブロック1/ブロック2のパックされた部分，ブロック2の残りの部分/
ブロック3のパックされた部分，ブロック3の残りの部分/…
パックされたブロック99/ブロック100のパックされた部分，ブロック100の残りの部分

直接ビットコード化(1バイトワード)								
ケース	ワード値の範囲	減数	バランス	コード化するための必要ビット数	制御ワード	コード化されたビット+制御ワードのビット	分析ビット	コード化されたビット+制御ワード+分析ビット
1	256 > >127	$2^8 \div 2 = 128$	0 - 127	7	0	8	X	8
2	128 > > 63	$2^8 \div 4 = 64$	0 - 63	6	1	7	0	8
3	64 > > 31	$2^8 \div 8 = 32$	0 - 31	5	11	7	1	8
4	32 > > 15	$2^8 \div 16 = 16$	0 - 15	4	101	7	1	8
5	16 > > 7	$2^8 \div 32 = 8$	0 - 7	3	1001	7	1	8
6	8 > > 3	$2^8 \div 64 = 4$	0 - 3	2	10001	7	1	8
7	4 > > 1	$2^8 \div 128 = 2$	0 - 1	1	100001	7	1	8
8	2 > > 0	$2^8 \div 256 = 1$	0	0	1000001	7	1	8
9	0	0	0	0	10000001	8	1	9
X = 不必要								

図35

特表平8-512438

例: STRING$ ⇒ バイト1 = 32_{10}
バイト2 = 255_{10}
バイト3 = 17_{10} ⇒ 図35の表より適当なワード値の範囲を見つけ、対応した減数を選択し、対応するバイトから減数を引く
バイト4 = 67_{10}
バイト5 = 203_{10}

			バランス(B) 7654321	制御ワード(CW) 8654321	分析ビット	ビットの長さ
ケース3: EB1$ ⇒	1 - 32	⇒	00000	11	1	8
ケース1: EB2$ ⇒	2 - 128	⇒	1111111	0	X	8
ケース4: EB3$ ⇒	3 - 16	⇒	0001	101	1	8
ケース2: EB4$ ⇒	4 - 64	⇒	00001.1	1	0	8
ケース1: EB5$ ⇒	5 - 128	⇒	1001011	0	X	8

B出力ＳＴＲＩＮＧ$(ＢＯＵＴＳ$)の作成 = EB1$(B) / EB2$(B) / EB3$(B) / EB4$(B) / EB5$(B)
CW/R出力ＳＴＲＩＮＧ$(ＣＷ/ＲＯＵＴＳ$)の作成= EB1$(CW) / EB2$(CW) / EB3$(CW) / EB4$(CW) / EB5(CW) / EB5$(R) / EB4$(R) / EB3$(R) / EB2$(R) / EB1$(R)

BOUTS$ = 00000 / 1111111 / 0001 / 000011 / 1001011
CW/ROUTS$ = 11 / 0 / 101 / 1 / 0 / X / 0 / 1 / X / 1

X = 分析ビット不必要

図 36

【図３７】

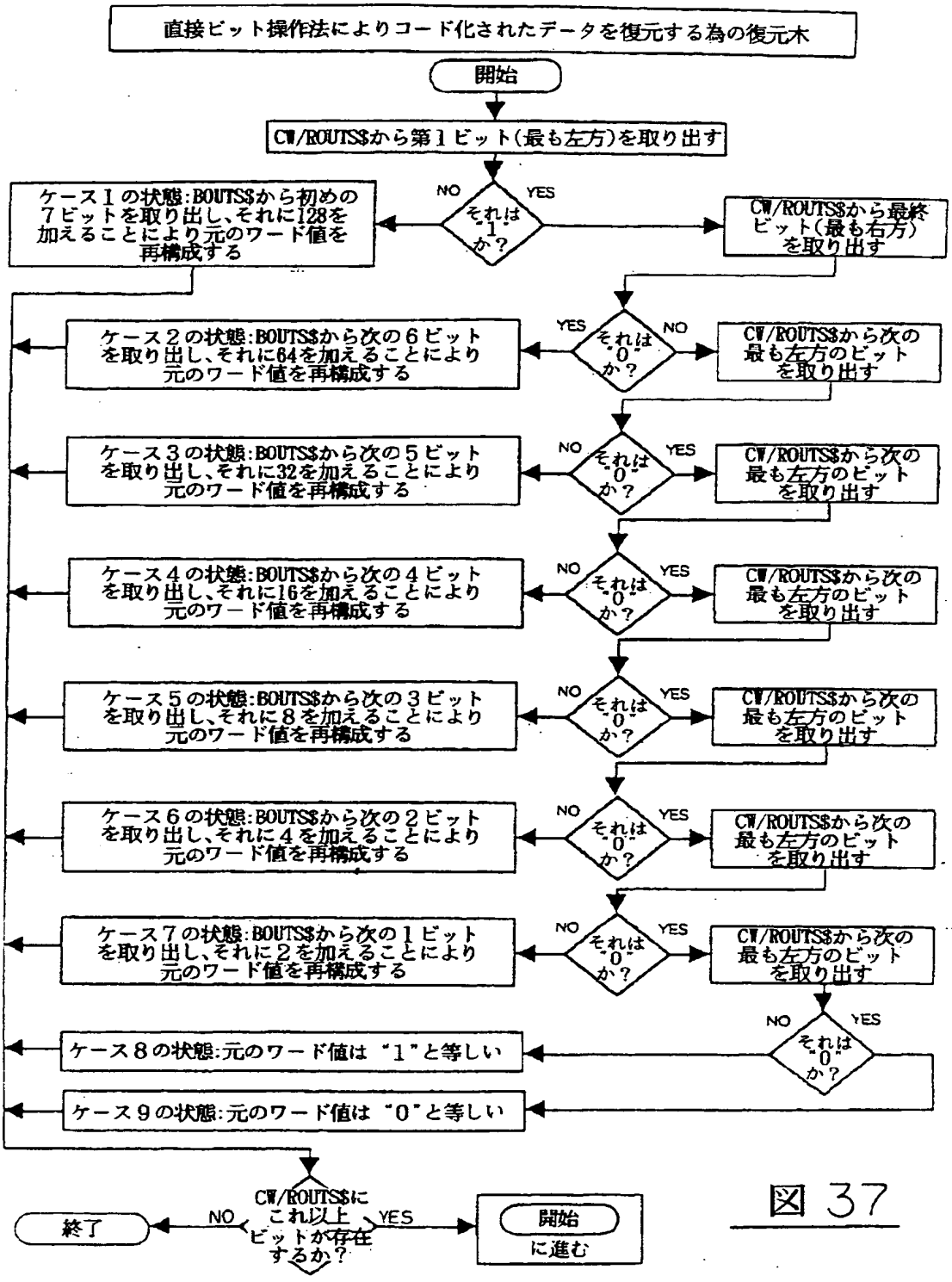

直接ビット操作法によりコード化されたデータを復元する為の復元木
開始
CW/ROUTS$から第１ビット(最も左方)を取り出す
それは"1"か？
NO
YES
ケース１の状態:BOUTS$から初めの７ビットを取り出し、それに128を加えることにより元のワード値を再構成する
CW/ROUTS$から最終ビット(最も右方)を取り出す
それは"0"か？
YES
NO
ケース２の状態:BOUTS$から次の６ビットを取り出し、それに64を加えることにより元のワード値を再構成する
CW/ROUTS$から次の最も左方のビットを取り出す
それは"0"か？
NO
YES
ケース３の状態:BOUTS$から次の５ビットを取り出し、それに32を加えることにより元のワード値を再構成する
CW/ROUTS$から次の最も左方のビットを取り出す
それは"0"か？
NO
YES
ケース４の状態:BOUTS$から次の４ビットを取り出し、それに16を加えることにより元のワード値を再構成する
CW/ROUTS$から次の最も左方のビットを取り出す
それは"0"か？
NO
YES
ケース５の状態:BOUTS$から次の３ビットを取り出し、それに８を加えることにより元のワード値を再構成する
CW/ROUTS$から次の最も左方のビットを取り出す
それは"0"か？
NO
YES
ケース６の状態:BOUTS$から次の２ビットを取り出し、それに４を加えることにより元のワード値を再構成する
CW/ROUTS$から次の最も左方のビットを取り出す
それは"0"か？
NO
YES
ケース７の状態:BOUTS$から次の１ビットを取り出し、それに２を加えることにより元のワード値を再構成する
CW/ROUTS$から次の最も左方のビットを取り出す
それは"0"か？
NO
YES
ケース８の状態:元のワード値は "1"と等しい
ケース９の状態:元のワード値は "0"と等しい
CW/ROUTS$にこれ以上ビットが存在するか？
NO
YES
終了
開始に進む

図 37

【図３８】

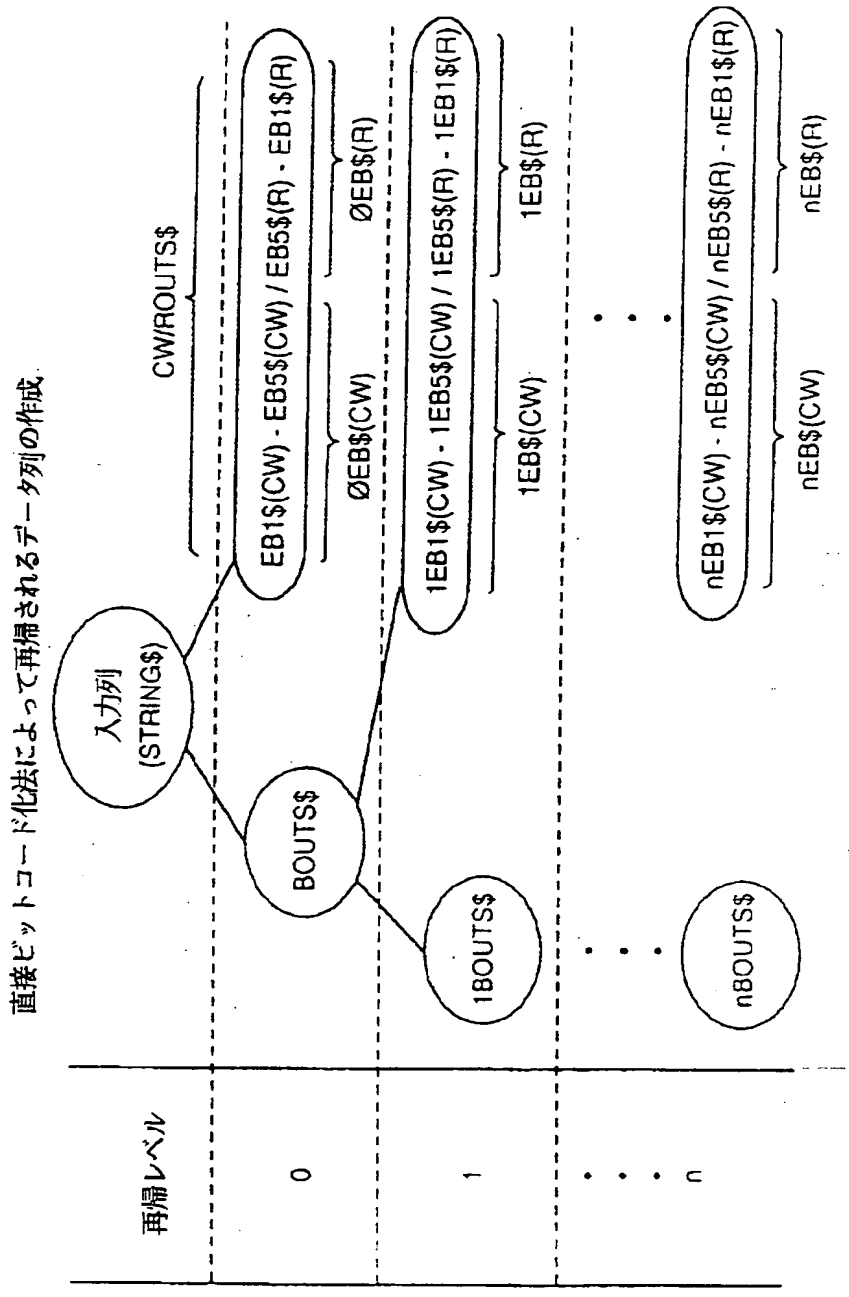

直接ビットコード化法によって再帰されるデータ列の作成
再帰レベル
0
1
n
入力列
(STRING$)
CW/ROUTS$
BOUTS$
EB1$(CW) - EB5$(CW) / EB5$(R) - EB1$(R)
ØEB$(CW)
ØEB$(R)
1BOUTS$
1EB1$(CW) - 1EB5$(CW) / 1EB5$(R) - 1EB1$(R)
1EB$(CW)
1EB$(R)
nBOUTS$
nEB1$(CW) - nEB5$(CW) / nEB5$(R) - nEB1$(R)
nEB$(CW)
nEB$(R)

図 38A

【図３８】

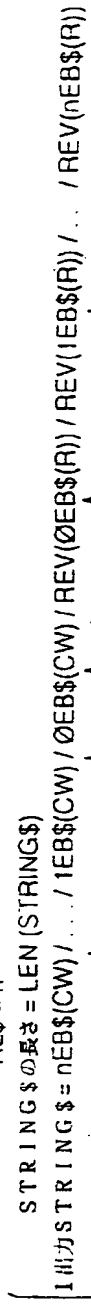


図38B

図38C

直接ビットコード化(２バイトワード)								
ケース	ワード値の範囲	減数	バランス	バランス値をコード化するための必要ビット数	制御ワード	コード化されたビット＋制御ワードのビット	分析ビット	コード化されたビット＋制御ワード＋分析ビット
1	$2^{16} \div 1 > > (2^{16} \div 2) - 1$	$2^{16} \div 2$	$0 - (2^{16} \div 2) - 1$	15	0	16	X	16
2	$2^{16} \div 2 > > (2^{16} \div 4) - 1$	$2^{16} \div 4$	$0 - (2^{16} \div 4) - 1$	14	1	15	0	16
3	$2^{16} \div 4 > > (2^{16} \div 8) - 1$	$2^{16} \div 8$	$0 - (2^{16} \div 8) - 1$	13	11	15	1	16
・	・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・	・
15	$2^{16} \div 2^{14} > > (2^{16} \div 2^{15}) - 1$	$2^{16} \div 2^{15}$	$0 - (2^{16} \div 2^{15}) - 1$	1	10000000000001	15	1	16
16	$2^{16} \div 2^{15} > > (2^{16} \div 2^{16}) - 1$	$2^{16} \div 2^{16}$	0	0	100000000000001	15	1	16
17	0	0	0	0	1000000000000001	16	1	17
X = 不必要								

図39

【国際調査報告】

INTERNATIONAL SEARCH REPORT

In..national application No.
PCT/US94/02088

A. CLASSIFICATION OF SUBJECT MATTER
IPC(5) :Please See Extra Sheet.
US CL :Please See Extra Sheet.
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED
Minimum documentation searched (classification system followed by classification symbols)
U.S. : 375/122; 364/900; 341/50, 51, 55, 60, 64, 65, 67, 79, 87, 90, 95, 106, 107; 382/40, 56

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US, A, 4,881,075 (WENG et al) 14 November 1989.	1-38
A	US, A, 4,955,066 (NOTENBOOM) 04 September 1990.	1-38
A	US, A, 4,782,325 (JEPPSSON et al) 01 November 1988.	1-38
A	US, A, 4,796,003 (BENTLEY et al) 03 January 1989.	1-38
A	US, A, 4,672,539 (GOERTZEL) 09 June 1987.	1-38
A	US, A, 4,597,057 (SNOW) 24 June 1986.	1-38
A	US, A, 4,545,032 (MAK) 01 October 1985.	1-38

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later documents published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 06 JUNE 1994	Date of mailing of the international search report JUL 11 1994
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer EDWARD L. COLES, SR. Telephone No. (703) 305-4712

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/02088

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US, A, 4,491,934 (HEINZ) 01 January 1985.	1-38
A	US, A, 3,694,813 (LOH et al) 26 September 1972.	1-38

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/02088

A. CLASSIFICATION OF SUBJECT MATTER:
IPC (5):

H04B 1/66; G06F 1/00; H03M 7/00, 7/34, 7/38, 5/00, 7/00, 7/40; G06K 9/36, 9/46

A. CLASSIFICATION OF SUBJECT MATTER:
US CL :

375/122; 364/900; 341/50, 51, 55, 60, 64, 65, 67, 79, 95, 106, 107; 382/56

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

APS
search terms: data compression, nibble encryption, nibble encoding, concatenating, parsing, direct bit manipulation, redundant data, packing, distribution compression